1997

An application of rough sets to economic and stock market data / c by Joseph A. Tremba

Joseph A. Tremba
San Jose State University

AN APPLICATION OF ROUGH SETS TO ECONOMIC AND STOCK MARKET DATA

A Thesis

Presented to

The Faculty of the Department of Mathematics and Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements of the Degree

Master of Science

by

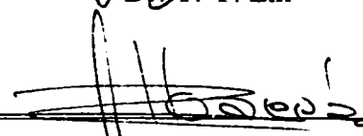
Joseph A. Tremba

December 1997

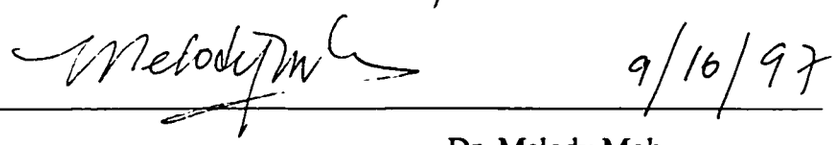
APPROVED FOR THE DEPARTMENT OF

MATHEMATICS AND COMPUTER SCIENCE

Dr. T. Y. Lin

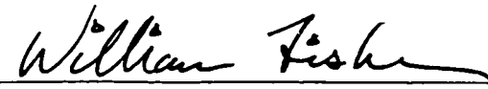


Dr. Mario Albarran



Dr. Melody Moh

APPROVED FOR THE UNIVERSITY



ABSTRACT

AN APPLICATION OF ROUGH SETS TO ECONOMIC AND STOCK MARKET DATA

by Joseph A. Tremba

This thesis explores the use of a relatively new field in the artificial intelligence area called Rough Set Theory to the application of economic and stock market data. The theory of Rough Sets is reviewed along with alternative AI methods. A learning experiment is set up using representative condition attributes from the computer, semiconductor, and semiconductor equipment industries. The program DataLogic/R+ is used as the rule generator. Computer programs are written to preprocess and discretize the time series data. The effects of averaging, delaying, and summing are explored. The propositional rules generated are then validated using recent data. The results show that strong predictive rules can be generated.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Making machines intelligent has been a goal for many years, perhaps dating back to the invention of the ENIAC computer in the 1940's. The more recent promises of Artificial Intelligence (AI) have not materialized even with the development of today's modern high speed computers. The promise of the 4GL language that would allow Japan to leap frog the United States was a failure. The technical challenges have proven to be more difficult than anticipated. Slower but more steady progress has been made. The recent innovations of Neural Networks, Genetic Algorithms, Fuzzy Logic, and Rough Sets provide new tools for researchers.

The thesis is organized as follows. Related AI methods are briefly discussed first. Then rough set theory and its components are presented. The methods available for predicting stock market performance are next. The semiconductor industry is used as the focus of the rough-set experiments. Attribute selection and data preparation methods are then presented. Next the rules generated from the learning experiment are listed. The validation of the rules is covered in section 8, followed by the conclusions.

2 Artificial Intelligence Methods

The methods available in the artificial intelligence field have broadened in recent years. The methods relevant to this thesis are neural networks, genetic algorithms, fuzzy logic,

and rough sets. They offer an alternative to the traditional statistical method known as the Bayesian Approach.

2.1 Bayesian Approach

The Bayesian approach uses statistics to determine the rules. Bayes Theorem is a means of calculating conditional probability [1].

$$P(H_i|E) = \frac{P(E|H_i)\cdot P(H_i)}{\sum_{k=1}^{n}\left(P(E|H_k)\cdot P(H_k)\right)}$$ where $P(H_i)|E$ is the probability that H_i is true given E. Typically a Gaussian distribution of events or attributes is assumed. The approach is based on the independence of attributes which cannot be guaranteed apriori. Hence the results are not always satisfactory. It is used quite often because it is easy to implement.

2.2 Neural Networks

Neural networks attempt to duplicate the working of the human brain.

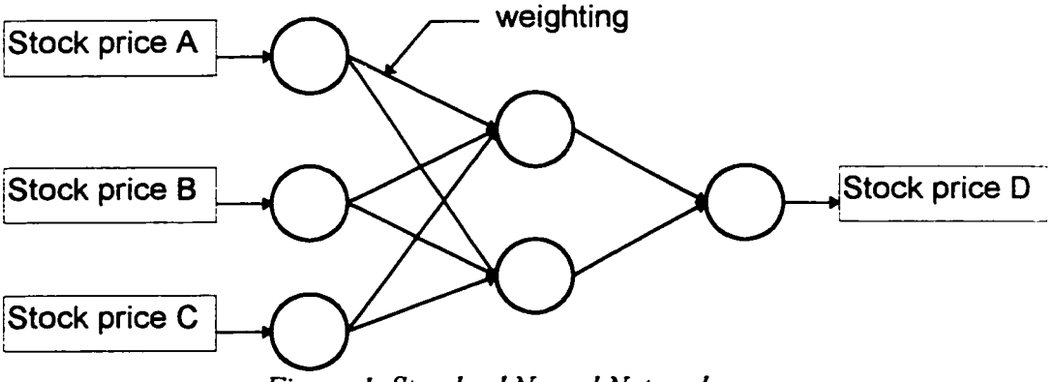


Figure 1. Standard Neural Network

In Figure 1 the stock prices for three stocks A, B, and C are the input and the price of stock D is the output. In this simplified explanation, the weighting of each input node is adjusted to obtain the desired output. The model is "trained" by using known examples until the output can be represented accurately for each set of inputs. The model is then applied in a predictive mode with a new set of input data to predict the stock price of D. This approach is successful and may presently be the most popular method [32]. Neural networks have two disadvantages. First, there are no rules generated. The user can not gain an intuitive feeling for the output of the model. The network is essentially a black box. Second, the training time is usually very long because the number of cases or objects required is large to obtain the desired accuracy.

2.3 Genetic Algorithms

A genetic algorithm approach is based on the evolutionary process of animals where the propagation of desired traits happens by natural selection. In each generation the best traits are combined to produce offspring that are better than their parents. This is a greedy algorithm. The process might be applied to the stock market as follows. First a population of candidate rules is assembled into subsets of 50 - 100 "organisms" [12]. They are combined using a crossover operation producing offspring with the same cardinality. For each group the best rules are chosen for the offspring. This process iterates for many generations. Mutations are also permitted. The genetic algorithm approach appears to work best on smaller databases [12]. It also has a similar drawback to neural networks in that it appears like a black box [2].

2.4 Fuzzy Logic

Fuzzy Logic is useful in dealing with uncertainty and vagueness. It has been used to predict economic events [2]. A membership function is defined to assign degrees to vague or subjective descriptions such as "high" or "low" inflation. One disadvantage of this approach is that the membership function is not objectively defined, but requires a domain expert.

2.5 Rough Set Approach

The rough set approach offers some advantages over the three previous approaches. First, it makes no assumption about the independence of the attributes or objects in the decision table. Second, the output is rules in propositional logic form that may be evaluated by a domain expert. Third, the rules may be generated from examples using known algorithms and do not require apriori knowledge or a domain expert.

2.6 Combination Approaches

The above approaches should not be viewed as being in competition but rather as complementing each other. Each approach has advantages and disadvantages. Recently it has been shown that combining two or more of the approaches can produce stronger results [32] [36]. Typically a rough set technique has granularity. If a more precise result is desired, then a combination of rough set preprocessing and a neural network backend

[15] or a genetic algorithm backend [36] has produced better results. In large databases, rough sets can reduce superfluous data [32].

3 Rough Set Theory

Classical set theory deals with crisp or deterministic sets. It cannot handle vagueness or uncertainty. Rough set theory is a way to gain knowledge or information from data that contains uncertainty [27]. Rough set theory, which was invented by Zdzislaw Pawlak, is based on equivalence relations and partitioning of finite sets. Recall that given a set S, a binary relation R on S is a subset of $S \times S$. The relation or predicate may have the properties of being reflexive, symmetric, or transitive which were defined as follows [6]:

R is reflexive: $\quad x \in S \rightarrow (x, x) \in R$

R is symmetric: $\quad (x, y \in S \wedge (x, y) \in R) \rightarrow (y, x) \in R$

R is transitive: $\quad (x, y, z \in S \wedge (x, y) \in R \wedge (y, z) \in R) \rightarrow (x, z) \in R$

An equivalence relation is a binary relation on a set that is reflexive, symmetric, and transitive. Further, an equivalence relation R partitions the set S into a collection of nonempty disjoint subsets of S whose union equals S. The set $[x]_R$ is called the equivalence class of x and contains all members of S related to x by R. This can be stated $[x]_R = \{y \mid y \in S \wedge xRy\}$. The quotient set, denoted S/R, is the set of all equivalence classes of R.

Given a finite and non-empty set U called the universe and an equivalence relation $R \subseteq U \times U$. A Pawlak approximation space is defined as the pair (U, R). For an arbitrary set $X \subseteq U$, it may not be possible to describe X precisely in terms of information in (U, R). Pawlak defined two approximations for describing X called the lower and upper approximations. The lower approximation, which is called $\underline{R}$, contains those objects that are known for certain to be in X. It can be expressed in the following equivalent statements [38]:

$$\underline{R}(X) = \{x \in U \mid [x]_R \subseteq X\}$$

$$\underline{R}(X) = \{x \in U \mid \textit{for all } y \in U, xRy \textit{ implies } y \in X\}$$

$$\underline{R}(X) = \cup\{[x]_R \mid [x]_R \in U / R, [x]_R \subseteq X\}$$

Lin [20] and Yao [38] point out that the equivalence class $[x]_R$ can have a dual role. First, it is a subset of U if viewed with respect to the universe; it will be denoted by $[x]_R$. Second, it is an element of the quotient set U/R; it will be denoted by the name $[x]_R$. If the role is clear from its context, $[x]_R$ may be used.

The upper approximation called $\overline{R}$ contains those objects that may belong to X. It can be given by the equivalent expressions:

$$\overline{R}(X) = \{x \in U \mid [x]_R \cap X \neq \varnothing\}$$

$$\overline{R}(X) = \{x \in U \mid \textit{there exists a } y \in U, \textit{such that } xRy \textit{ and } y \in X\}$$

$$\overline{R}(X) = \cup\{[x]_R \mid [x]_R \in U / R, [x]_R \cap X \neq \varnothing\}$$

A graphical example will help illustrate the two approximations. Let there be a universe U that contains many elements that are of two types A and B called attributes. Attribute A has the range of integer numbers with values $0 \leq A \leq 8$ and attribute B has the integer value range of $0 \leq B \leq 7$. Let the universe be partitioned into 7 x 8 = 56 equivalence classes using the simple equivalence relation that A and B have unique integer values. Now let X be a set bounded by an ellipse. The lower approximation of X is shown in Figure 2. In equation form it is $\underline{R}(X) = \{[3,4],[4,3],[4,4],[4,5],[5,4]\}$ where $[a,b]$ is the equivalence class of the attribute pair. Similarly the upper approximation of X is shown in Figure 3 and contains those classes that may belong to X. It is equal to

$$\overline{R}(X) = \{[2,3],[2,4],[2,5],[3,2],[3,3],[3,4],[3,5],[3,6],[4,2],\cdots,[6,3],[6,4],[6,5]\}$$

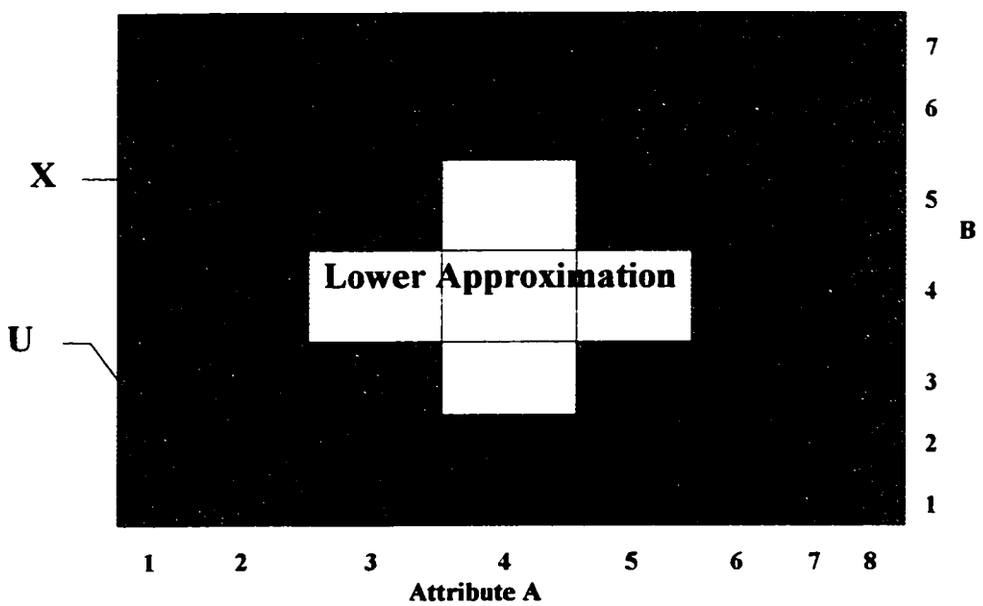


Figure 2. Lower Approximation Region

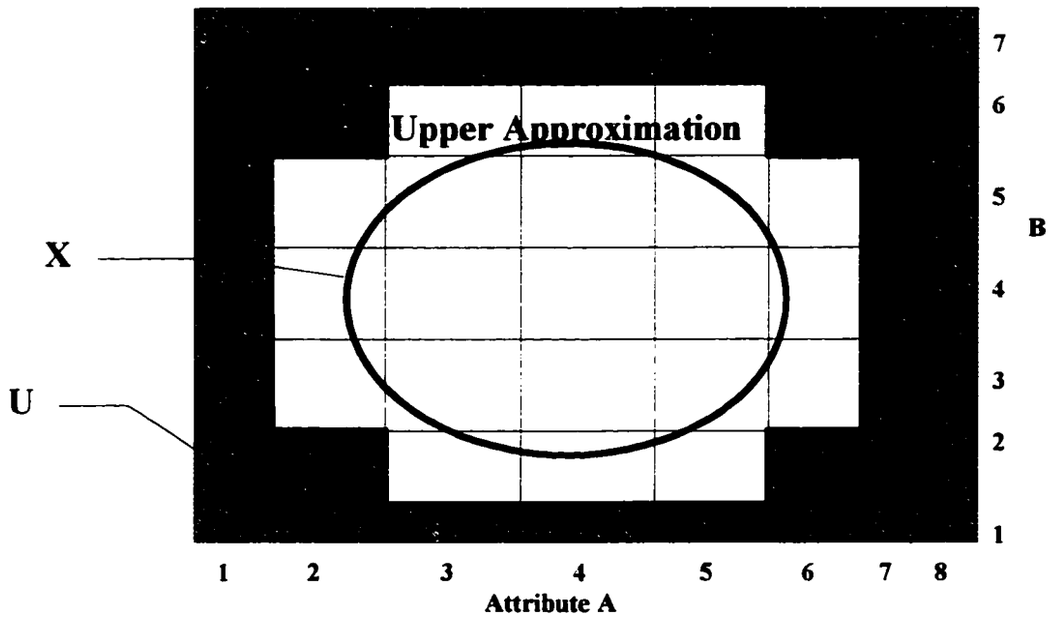


Figure 3. Upper Approximation Region

An important observation can also be made from this simple example. First, the attribute *A* partitions the universe into vertical strips. Hence it is an equivalence relation [6]. Likewise attribute *B* partitions the universe and is an equivalence relation. The 56 equivalence classes are the intersection of the two partitions over the whole plane. This illustrates an elementary property, namely, that the intersection of two equivalence relations is another equivalence relation consisting of all possible intersections of the equivalence classes of the two equivalence relations as stated by Lin [16].

Pawlak goes on further to define three regions which are the positive region $POS(X)$, the negative region $NEG(X)$, and the boundary region $BND(X)$. The positive region is the same as the lower approximation. The negative region is everything in the universe that is

not in the upper approximation. Lastly, the boundary region is the difference between the upper and lower approximations. Expressed as equations the three regions are:

$$POS(X) = \underline{R}(X)$$

$$NEG(X) = U - \overline{R}(X)$$

$$BND(X) = \overline{R}(X) - \underline{R}(X)$$

Graphically, the three regions are shown in Figure 4. A variation of Figure 4 has traditionally been used to illustrate rough set regions [11][17] [32] [37]. Rough set theory is about the study of the boundary region [26]. If there is no uncertainty, then $\underline{R}X = \overline{R}X$ and $BND(X) = \varnothing$. Rough set theory is the study of uncertainty when the uncertainty is described by equivalence relations. The sources of uncertainty are from three sources generally which are noise, missing data, and errors [10]. The sources of uncertainty are not discussed in this thesis.

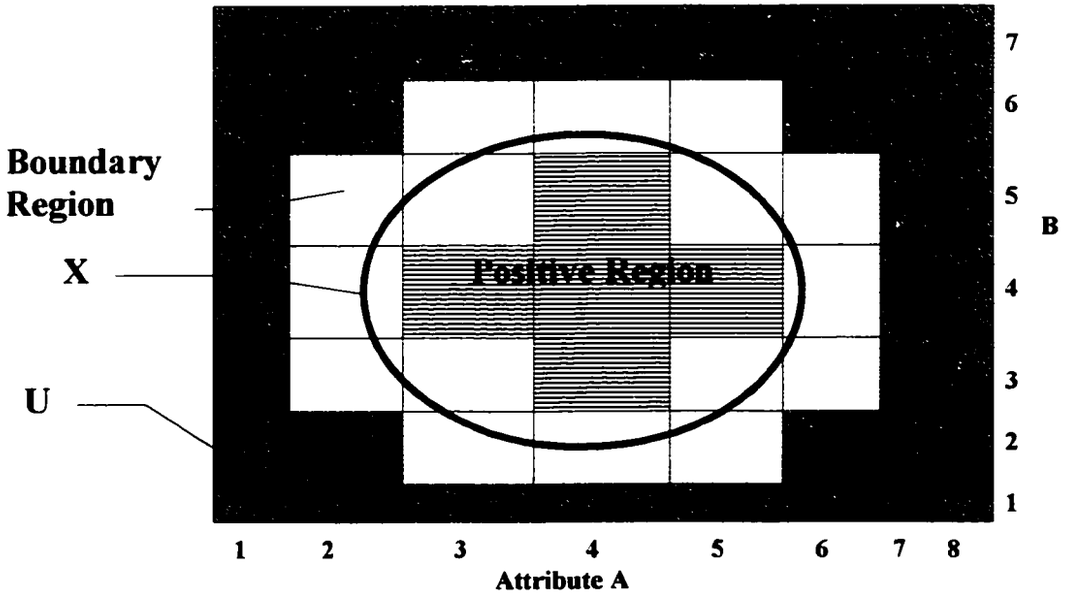


Figure 4. Rough Set Regions

Pawlak [25] also proves the following twelve propositions about the lower and upper approximations:

1. $\underline{R}X \subseteq X \subseteq \overline{R}X$
2. $\underline{R}\varnothing = \overline{R}\varnothing = \varnothing$ and $\underline{R}U = \overline{R}U = U$
3. $\overline{R}(X \cup Y) = \overline{R}X \cup \overline{R}Y$
4. $\underline{R}(X \cap Y) = \underline{R}X \cap \underline{R}Y$
5. $X \subseteq Y \rightarrow \underline{R}X \subseteq \underline{R}Y$
6. $X \subseteq Y \rightarrow \overline{R}X \subseteq \overline{R}Y$
7. $\underline{R}(X \cup Y) \supseteq \underline{R}X \cup \underline{R}Y$
8. $\overline{R}(X \cap Y) \subseteq \overline{R}X \cap \overline{R}Y$
9. $\underline{R}(-X) = -\overline{R}X$
10. $\overline{R}(-X) = -\underline{R}X$
11. $\overline{R}(\overline{R}(X)) = \overline{R}(X)$
12. $\underline{R}(\underline{R}(X)) = \underline{R}(X)$

Some of the propositions are so strong that they can characterize the equivalence relations. It can be shown [22] that for two abstract operators H and L from power sets $P(U)$ to $P(U)$ such that they satisfy 1, 2, 3, 4, 11, and 12, then there is an equivalence relation R such that $H = \overline{R}$ and $L = \underline{R}$.

3.1 Decision Table

In the previous section the example shown in Figure 4 used an equivalence relation formed by the intersection of two equivalence relations to form the lower and upper approximations. The equivalence relation formed a quotient set and partitioned the universe. The values of the individual attributes are equivalence classes. The mapping to the attribute values is a function. Expanding on this, an information table or information system can be introduced. Let the number of attributes be a finite number and represented by the set A. Also let V be a set which is the union of the set of values that each attribute can have. Let there be a function ρ that maps each entry in the table to a value for each attribute. An information table has been induced. An information table has been formally defined [17] as $S = (U, A, V, \rho)$ where

U is a nonempty set of objects,

A is a nonempty set of attributes,

V is a nonempty union of all sets of values V_a for each attribute $a \in A$,

$\rho : U \times T \rightarrow V$, called a description function, is a mapping such that

$\rho(x, a)$ *is in* V_a *for all* x *in* U *and* a *in* V

Information tables, also known as information systems or knowledge representation systems, are representations of knowledge using names of equivalence classes. Pawlak [25] shows that a universe with n equivalence relations, called Pawlak knowledge base, can be represented by an information table and vice versa.

Theorem: There is a one-to-one correspondence between Pawlak knowledge bases and information tables.

The equivalence of an information table and a knowledge base has been confirmed by Pawlak [25], Lin [16], and Yao [37].

An information table is a table that consists of rows and columns. The columns are attributes and the rows are objects or instances. Duplicate objects are permitted, unlike a relational database that does not allow duplicates. An additional refinement is needed for the stock market application. The information table must also be a decision table. A decision table is a special form of information table where the attribute set $A = C \cup D$ and $C \cap D = \varnothing$. C is the set of condition attributes and D is the set of decision attributes. For the experiments in this thesis, $D = \{d\}$ is a single decision attribute.

3.2 Concept Decision Table

Pawlak defined a subset of the universe of discourse as a concept [25]. Intuitively a class of an object represents some concept. For example, given a set of balls, they may be grouped according to their weight. Divide the group of balls into two sets X and Y where $X = \{balls\, weighing < 5\, ounces\}$ and $Y = \{balls\, weighing > 2\, pounds\}$. According to Pawlak, X represents some concept in the universe (of balls). For convenience, this set is given a name. A meaningful name such as light would be suggestive e.g. $name(X) = light$. Similarly for the set Y, $name(Y) = heavy$. In other words, a concept is a name of a subset of attribute values that intuitively is some knowledge derived from the set of attribute values. Assume there is a partition on the set of decision attribute values. Then the name of each equivalence class represents a concept. A concept decision table is

derived from a decision table by replacing each attribute value x of decision attribute by the $name([x])$, where $[x]$ is the equivalence class containing x. A domain expert is needed to develop concepts in an application, i.e. partioning the equivalence class and assigning a meaning name. See Lin [18] [19] [21]. With the use of concepts, the goal would be to derive strong rules. Strong rules are rules that are supported by a large number of cases. In this stock market experiment it was necessary to operate on the raw data to provide the predictive quality. Prediction was obtained by comparing present data with past data. Also the closing stock price was converted to percentage change to remove absolute values. Averages and sums were used to smooth the data. Lastly the decision attribute values were mapped to three concepts, which were rising, falling, and no change. The condition attributes could have been mapped to three values also. They were not changed because strong rules were obtainable by rounding the percentage change to the nearest integer.

3.3 Indiscernibility

In a decision table, two or more objects may have the same values for some subset $B \subseteq A$ of attributes. These objects that cannot be distinguished one from another using only the attributes B are called indiscernible. When they are indiscernible, they are members of the same equivalence class. That is $xR_B y \Leftrightarrow (\forall b \in B)\, \rho(x,b) = \rho(y,b)$ [37]. This is often denoted $IND(B)$. The indiscernibility is the essence of rough set theory.

Using *IND*( ), there is an alternate way of expressing the approximation sets as given by Pawlak [25] is:

$$\underline{R}X = \bigcup\{Y \in U / R : Y \subseteq X\}$$

$$\overline{R}X = \bigcup\{Y \in U / R : Y \cap X\} \text{ where } R \in IND(K) \text{ in } K = (U, R).$$

3.4 Reducts

Unnecessary, dispensable, or superfluous attributes can be removed. Pawlak [25] states that an attribute $a \in B$ is dispensable in $B \subseteq A$ if $IND(B) = IND(B - \{a\})$. Otherwise, it is indispensable in B. If B and C are a collection or family of sets and $C \subseteq B$ where the elements $a \in B - C$ are dispensable, then C is a reduct of B called *RED(B)* if $IND(C) = IND(B)$. Further the set of all indispensable relations in B is called the *CORE(B)* [17]. Deogun et al [5] state that the search for the minimum size reduct is exponential in the number of attributes and it might not always be feasible to do in large databases.

3.5 Probabilistic Rough Sets

The analysis of uncertainty has attempted to quantify the amount of uncertainty. This has led to the development of probabilistic rough sets [14] [37] [40]. Two probabilistic

approaches are Rough Membership Function (RMF) and Variable Precision Rough Sets (VPRS).

3.5.1 Rough Membership Function

Pawlak [25] defines a probability function for the elements of the same equivalence class $[x]$ belonging to X as:

$$\mu_X(x) = \frac{|X \cap [x]_{\Re}|}{|[x]_{\Re}|} \quad \text{where } \Re \text{ is a collection of equivalence relations}$$

All members or elements of the equivalence have the same probability.

3.5.2 Variable Precision Rough Sets

The VPRS model developed by Ziarko [40] generalizes the probability function into the lower and upper approximation functions and into the three regions based on a term $c(X,Y)$ called the misclassification measure which is defined as:

$$c(X,Y) = 1 - \frac{|X \cap Y|}{|X|} \qquad \text{if } |X| > 0 \text{ or}$$

$$c(X,Y) = 0 \qquad \text{if } |X| = 0$$

Another term β is called the admissible classification error and $0 \le \beta < 0.5$. Then

$$\underline{R}_\beta X = \cup\{E \in \Re : c(E, X) \le \beta\} \text{ and}$$

$$\overline{R}_\beta X = \cup\{E \in \Re : c(E, X) < 1 - \beta\}$$

Similar definitions are made for $POS_{\beta}(x)$, $BND_{\beta}(x)$, and $NEG_{\beta}(x)$. If $\beta = 0$ then the VPRS equations reduce to the Pawlak definitions. The importance of the VPRS approach is that it is used in the rule generation program DataLogic/R+ [39] in a later section. Current research is expanding on the VPRS approach into a Generalized Rough Set model [14].

3.6 Rules

The desired knowledge output of the computer program is a set of rules. They are best represented by propositional logic. Such logical formalism is called decision logic [25]. It will be used informally here. The rules consist of attribute-value pairs combined using conjunction, disjunction, and implication operators [16]. Examples are:

$$(stockA,2) \wedge (stockB,1) \rightarrow (stockC,3)$$

$$(stockA = 2) \wedge ((stockB = 3) \vee (stockD = 1)) \rightarrow (stockC = 3)$$

if stock A = 2 and either stock B = 3 or stock D = 1 then stock C = 3.

3.7 Discernibility Matrix and Function

An alternate method from Pawlak's method of reducing a decision table to reducts was proposed by Skowron and uses his discernibility matrix and discernibility function [31].

Recall the information system $S = (U, A, V, \rho)$. Let the attributes be $A = \{a_1, a_2, \ldots, a_m\}$ and the objects be $U = \{x_1, x_2, \ldots, x_n\}$. Let $M(S)$ be called the discernibility matrix which is an $n \times n$ matrix with elements (c_{ij}) where

$$c_{ij} = \{a \in A : a(x_i) \neq a(x_j)\} \text{ for } i, j = 1,2,\ldots,n$$

$M(S)$ is a symmetric matrix and $c_{ii} = \varnothing$ for $i = 1,2,\ldots,n$. It is typically shown in the lower triangle form since the upper part is the same as the lower part. The discernibility matrix is equivalent to the Pawlak approach.

The discernibility function called f_S is a Boolean function consisting of the Boolean variables $\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_m$ which correspond to the attributes $a_1, a_2, \ldots, a_m$ respectively. The function is defined as:

$$f_S(\bar{a}_1, \ldots, \bar{a}_m) = \wedge\{\vee(c_{ij}) : 1 \leq j < i \leq n, c_{ij} \neq \varnothing\}$$

where $\vee(c_{ij})$ is the disjunction of all variables $\bar{a}$ such that $a \in c_{ij}$. The discernibility function performs a conjunction of all the nonempty elements as Boolean variables of the discernibility matrix.

The function can be simplified easily using Boolean tautologies and the absorption property $x \wedge (x \vee y) = x$. The significance of this function is that it will produce reducts efficiently. From reducts, rules may be produced. The use of the discernibility matrix and discernibility function to generate rules will be shown in the next section.

3.8 Rule Simplification

The rules produced using the discernibility matrix and discernibility function are not necessarily minimum. The rules generated are based on applying the reducts to the decision table. Further simplification may be possible by removing attributes from some of the rules. Pawlak calls this process "value reducts" [25]. The steps involved start by examining a subset of the condition attributes in a rule. For each rule, if the values of the condition attribute subset uniquely define the decision value, then the rule may be reduced to use only the subset of attributes.

3.9 Rule Generation Algorithm

Skowron [31] shows that the discernibility matrix and function are efficient and can be computed in polynomial time $O(n^2)$. An algorithm then can be developed that will produce reducts in $O(n^2)$ time. The reducts produced however are not unique. Nor are they minimal. Skowron also shows that finding an algorithm to produce minimal reducts or Minimal Reduct Problem is NP-Hard [31]. A work-around method for this difficulty uses heuristics to minimize the reducts. Figure 5 is an algorithm for generating rules using the discernibility matrix and discernibility function.

1. Remove duplicate attributes and objects.
2. Construct the relative discernibility matrix. Note inconsistent objects.
3. Determine the discernibility function.
4. Minimize discernibility function using absorption property. This produces relative reducts, but not necessarily minimal reducts.
5. Identify objects containing unique condition-attribute values from discernibility matrix.
6. Convert objects with unique attribute values to rules.
7. Convert remaining objects to rules using reducts from step 4.
8. Minimize each rule by removing unnecessary attributes and combining rules.
9. Calculate cardinality for inconsistent rules.

Figure 5. Rule Generation Algorithm Using Discernibility Matrix

3.9.1 Algorithm Example Using Continuous Data

The following simple example will illustrate the rule generation algorithm applied to continuous data. It also shows how to discretize continuous data. The goal of the example will be to find rules between the price of two stocks A and B. The stock price is represented by sin functions as:

$$A = 2 + \sin x$$

$$B = 3 + \sin x \quad \text{where } 0 \le x < 2\pi$$

Rough set theory deals with finite sets. It cannot handle continuous data directly because continuous data contains an infinite number of points. A way around this is to discretize the data into finite points or ranges. For this example, let B have two values *plus* when $B > 3.0$ and *minus* when $B \leq 3.0$. Similarly A will have four values defined as a when $1 \leq A \leq 1.5$, b when $1.5 < A \leq 2.0$, c when $2.0 < A \leq 2.5$, and d when $2.5 < A \leq 3.0$. A sample decision table is created with 13 objects. The objects of the decision table will be found by taking samples every 0.5 radians starting at 0. Table 1 is the starting decision table:

	Stock A	Stock B
1	b	minus
2	c	plus
3	d	plus
4	d	plus
5	d	plus
6	d	plus
7	c	plus
8	b	minus
9	a	minus
10	a	minus
11	a	minus
12	a	minus
13	b	minus

Table 1. Discretized Continuous-Data Decision Table

A graph of the decision table is shown in Figure 6 below. The data points are marked.

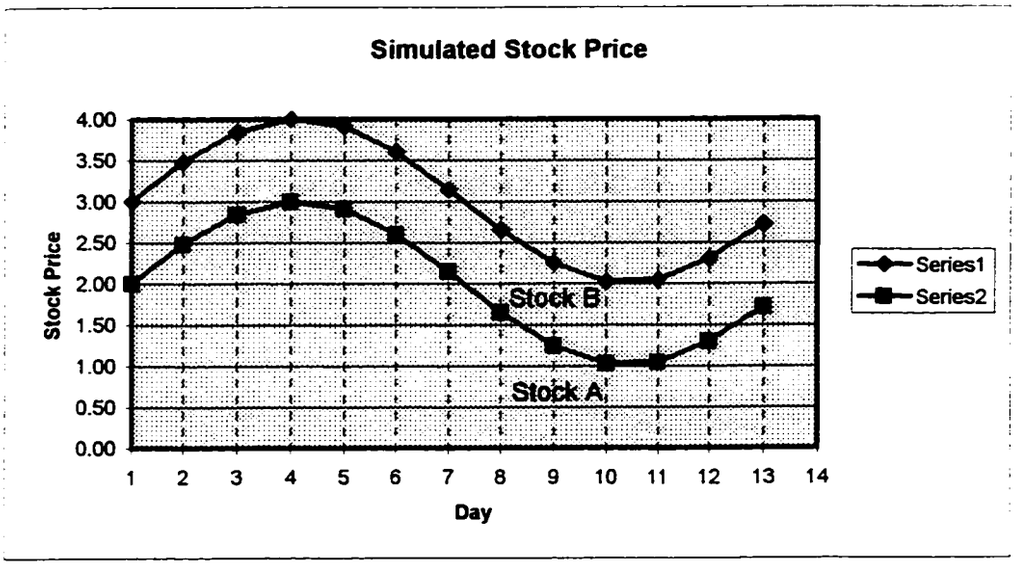


Figure 6. Graph of Simulated Stock Price

Step 1. There are several duplicate objects or rows in the table. The duplicate objects are {1, 8, 13}, {2}, {3,4,5,6}, and {9,10,11,12}. Removing the duplicates yields the table below.

	Stock A	Stock B
1	b	minus
2	c	plus
3	d	plus
9	a	minus

Table 2. Reduced Continuous-Data Decision Table

Step 2. When constructing the relative discernibility matrix, objects or rows with the same decision value are not compared. Instead a dashed line is entered in the matrix cell. The discernibility matrix is:

	1	2	3	4
1				
2	A			
3	A	----		
4	----	A	A	

Table 3. Discernibility Matrix for Continuous Data Example

Step 3. - 5. The discernibility function is $f_S = A$. The corresponding reduct is $RED(S) = \{\{A\}\}$.

Step 6. - 7. The rules will have only the single attribute A in them. The attribute and decision values are taken from the decision table for forming the rules:

$A = b \rightarrow B = \min us$

$A = c \rightarrow B = plus$

$A = d \rightarrow B = plus$

$A = a \rightarrow B = \min us$

Step 8. Converting back to the continuous ranges and combining rules gives:

$1.0 \leq A \leq 2.0 \rightarrow B \leq 3.0$

$2.0 < A \leq 3.0 \rightarrow B > 3.0$

Step 9. This step is not needed because the deterministic function has no inconsistent rules.

When viewing the discretization of the continuous data in an abstract sense, it can be considered a generalization. Each of the values are replaced by an interval. If a name is given to each interval, a high level concept is formed. This is an example of the idea behind the concept decision table described earlier.

3.9.2 Algorithm Example with Inconsistent Data

Another example of this algorithm will use the same decision table presented by Grzymala-Busse [9]. The table contains inconsistent data.

	Condition				Decision
	Temperature	Hemoglobin	Blood Pressure	Oxygen Saturation	Comfort
1	low	fair	low	fair	low
2	low	fair	normal	poor	low
3	normal	good	low	good	low
4	normal	good	low	good	medium
5	low	good	normal	good	medium
6	low	good	normal	fair	medium
7	normal	fair	normal	good	medium
8	normal	poor	high	good	very_low
9	high	good	high	fair	very_low

Table 4. Hypothermic Post Anesthesia Patients

Step 1. No duplicates attributes or objects were found.

Step 2. The relative decision matrix is shown in Table 5. Note that the attributes are not discerned within a decision equivalence class. This is consistent with Pawlak [25].

	1	2	3	4	5	6	7	8	9
1									
2	----								
3	----	----							
4	t,h,o	t,h,b,o	∅						
5	h,b,o	h,o	t,b	----					
6	h,b	h,o	t,b,o	----	----				
7	t,b,o	t,o	h,b	----	----	----			
8	t,h,b,o	t,h,b,o	h,b	h,b	t,h,b	t,h,b,o	h,b		
9	t,h,b	t,h,b,o	t,b,o	t,b,o	t,b,o	t,b	t,h,b,o	----	

Table 5. Discernibility Matrix for Hypothermic Example

The element $c_{4,3}$ is empty because this is a case of inconsistent data where the same condition attribute values have different decision values.

Step 3. The discernibility function is:

$$f_S = (h \vee b) \wedge (h \vee o) \wedge (t \vee o) \wedge (t \vee b) \wedge (h \vee b \vee o) \wedge (t \vee b \vee o) \wedge (t \vee h \vee o) \wedge (t \vee h \vee b) \wedge (t \vee h \vee b \vee o)$$

Step 4. Applying the absorption property:

$$f_S = (h \vee b) \wedge (h \vee o) \wedge (t \vee o) \wedge (t \vee b)$$

Rearranging terms:

$$f_S = (b \wedge o) \vee (t \wedge h)$$

The reducts are $RED(S) = \{\{b, o\}, \{t, h\}\}$

Step 5. Calls for identifying unique attribute values in objects. A unique value can be identified by looking at each matrix element for an object. If the attribute appears in all elements, then the attribute value is unique to that object. There are four such cases in this example. Object 2 has oxygen discernible from all other objects. Objects 3 and 9 have blood pressure unique. Object 8 has both hemoglobin and blood pressure unique.

Step 6. Rules can be made for the objects with unique values. The four rules are:

$oxygen = poor \rightarrow comfort = low$ object 2

$blood\_pressure = low \rightarrow comfort = low$ object 3

$blood\_pressure = high \rightarrow comfort = very\_low$ object 8,9

$hemoglobin = poor \rightarrow comfort = very\_low$ object 8

Step 7. Applying the two reducts will give two sets of rules. Using $\{b \wedge o\}$:

$blood\_pressure = low \wedge oxygen = fair \rightarrow comfort = low$ object 1

$blood\_pressure = low \wedge oxygen = good \rightarrow comfort = medium$ object 4

$blood\_pressure = normal \wedge oxygen = good \rightarrow comfort = medium$ object 5,7

$blood\_pressure = normal \wedge oxygen = fair \rightarrow comfort = medium$ object 6

Using reduct $\{t \wedge h\}$, the rules generated are:

$temperature = low \wedge hemoglobin = fair \rightarrow comfort = low$ object 1

$temperature = normal \wedge hemoglobin = good \rightarrow comfort = low$ object 4

$temperature = low \wedge hemoglobin = good \rightarrow comfort = medium$ object 5,6

$temperature = normal \wedge hemoglobin = fair \rightarrow comfort = medium$ object 7

Step 8. After examining the attribute, no further minimization was found.

Step 9. The discernibility matrix identified inconsistent data for objects 3 and 4. The rules generated for these two objects are:

$blood\_pressure = low \rightarrow comfort = low$ object 3

$blood\_pressure = low \wedge oxygen = good \rightarrow comfort = medium$ object 4

$temperature = normal \wedge hemoglobin = good \rightarrow comfort = low$ object 4

All three rules can be applied to either object 3 or 4 which give different comfort levels. The cardinality of all three rules is 0.5 because there is a 1 out of 2 choice.

The rules generated above were confirmed and listed in the Grzymala-Busse paper [9]. The paper lists other rules that may be generated from this table. This is to be expected because reducts are not unique, so different algorithms may generate different reducts and rules.

The algorithm was also tried on two decision tables from Pawlak's book [25]. One table had consistent data and the other had inconsistent data. The rules generated using the algorithm were validated against Pawlak's rules and found to be the same. In one case, additional minimal rules were found.

3.10 Example Programs

There are several knowledge discovery programs available that are based on rough set theory. They are LERS, DataLogic, RSES, KDD-R, and ROSETTA.

LERS, which is an acronym for Learning from Examples using Rough Set, was written by Grzymala-Busse [9]. It now has two modules LEM1 and LEM2 [8] where LEM stands for Learning from Example Module version 1 or 2. The versions provide options on how to classify rules induced from the boundary region. Two of the options are All Coverings and All Rules which provide a larger set of rules than generated by the machine learning options [5].

The program DataLogic/R+ is the one used in this thesis. It provides propositional rules from decision tables. It has two parameters that can be set by the user called roughness and precision [28]. Roughness can be varied between 0 and 1. Generally, high roughness results in a few, relatively strong rules. Strength here is a measure of the number of supporting cases for the rule. As roughness is decreased, more rules are generated but they typically have fewer supporting cases. The exact method used for determining roughness is proprietary, but Golan [7] states that it is based on the Variable Precision Rough Set (VPRS) model of Ziarko [40]. The rule precision parameter controls the minimum rule probability. When precision is decreased, more general rules are induced but less precise. If the precision is set very low it is possible to create the case where all rules are in the positive region. One draw back of the program is that it does not adjust the concept values automatically for continuous data. This means that the user via preprocessing must change the decision values. It would be better if the program would

do this automatically with the goal of obtaining the strongest rules with a complete covering.

The program RSES (Rough Set Expert System) was developed by the Logic Section in Warsaw University [2]. The newer versions of the program offer the option of scaling attribute values. The source code is made available on the world wide web and has been used for additional research [1].

The program KDD-R introduced in 1994 is based on the variable precision rough set model VPRS. Options are provided for discretization of the data by a user defined means or an internal algorithm. Superfluous attributes are eliminated [1][5].

ROSETTA is the newest program available in the research community. It is intended to be flexible so that new algorithms can be added and used easily. It is a toolkit that can use multiple selectable algorithms, such as RSES, on a Windows interface [24]. The kernel is a C++ library containing analysis and classification classes. The GUI is designed so that newly added algorithms can be integrated and displayed with a minimum of programming effort. The GUI and kernel are kept independent. The data input format is decision tables and the output is propositional rules. Ohrn indicated at the recent conference [24] that a limited version will be available in public domain on the web.

Other programs based on rough sets are RoughDAS by Slowinski and Stefanowski in 1992 and RSL by Gawrys and Seinkiewicz in 1994.

4 Predicting Stock Market Performance

One area that is extremely attractive for the application of AI is the stock market. The lure of wealth has been a motivater for many decades. The traditional analysis methods of fundamental analysis and technical analysis have provided useful short-term and long-term strategies, but no predictive guarantees. Some stock market followers believe in the random walk philosophy which theorizes that the day-to-day stock price is random. The following is a brief description of each approach.

Fundamental analysis involves looking at the details of the company, its management, and its related industry. The details considered include its annual report and reports from industry experts such as Value Line. Financial ratios such as price/earnings, return on assets, and profits/employee are examined and compared to normal guidelines. Performance over the last five years is tracked on sales growth and earnings growth for example. Criteria are set for each and applied to the database of stocks to filter out those stocks that best meet the criteria.

Technical analysis uses charting techniques to identify known patterns. The technician looks at the general market and price charts [3]. For the general market some examples of

parameters to be considered include market breadth (advances vs. declines), trading volume, odd-lot trading, and short selling. From stock price charts, the technician looks at moving averages, stochastic indicators, relative strength, and chart patterns such as head and shoulders pattern [35]. Based on past performance of known graphical patterns, buy or sell signals are derived.

Often a combination of fundamental and technical analysis is used. One such combination is to use fundamental analysis to determine what stock to buy, and then use technical analysis to determine when to buy it.

Detractors of the stock market site the random walk theory which states that the short-term price movement in the stock market is random. Any hope of predicting performance is more luck than skill [13]. Supporters of the random walk hypothesis believe that analysis of past data is not useful for predicting future prices because successive price changes are statistically independent [4]. Playing the stock market is like gambling. One advantage of gambling is that the odds are known for a given game. In the stock market, the odds are unknown. "Shallow men believe in luck, wise and strong men in cause and effect", said Ralph Waldo Emerson. The purpose of this thesis is to help define this cause and effect relationship.

The time span of investing also should be considered when investing. The two commonly used durations are short term and long term. Fundamental Analysis is often aligned to long term investing. Long term is usually considered to be from several months to several

years. Short term is usually considered less than six months. Short term investors are often called traders. Traders often use technical analysis and may buy and sell a stock within the same day.

4.1 Semiconductor Equipment Industry

The semiconductor equipment industry is a high-tech industry that developed to support the semiconductor industry. Both industries are relatively new and developed locally here in Northern California. Early in the development of semiconductors the equipment used to fabricate them was also designed and developed within the semiconductor companies. IBM, Motorola, and Texas Instruments had large equipment departments. Slowly, independent companies were started to design and manufacture the specialized equipment needed for this industry. This trend continued until today when all equipment is built by separate companies. The semiconductor companies specialized then in the design and manufacture of the semiconductor devices themselves. Applied Material is an equipment company that was founded about 30 years ago in Santa Clara, CA. Presently Applied Materials is the world leader in their industry with annual sales approaching four billion dollars.

There is a well known connection between companies in the semiconductor industry. This connection is often referred to as a food chain. This food chain starts with the general public as the end user. They buy personal computers. The semiconductor companies supply chips for computers. The semiconductor equipment companies supply capital

equipment to the semiconductor companies. The food chain connection also applies to the stock prices of these companies. If an economic upsurge is forecasted, people may buy more computers. The computer companies stock prices go up in anticipation of more profits. This may cause the price of semiconductor stocks to rise. In turn, the price of semiconductor equipment stocks may rise in anticipation of increased sales. In the past there was a time lag of as much as six months for these events to be observed. Today, in the information age, the lag is much smaller because pertinent knowledge is readily available sooner.

4.2 Reasons for Studying The Semiconductor Industry

The importance of an established relation between companies is two fold. First, the knowledge gathered should support this relation. If it does not, then there may be errors in the analysis or the program used. Rules that show a relation will validate accepted knowledge. Second, the performance of each of the related companies is not independent. Therefore any analysis that assumes independence would give erroneous results. Rough set theory makes no assumption about independence.

The semiconductor industry is a good candidate for studying because it is rich in data and activity. This industry is a global industry. Therefore it should be less sensitive to the economy of a single country. There are several indexes available that are used to track its performance e.g. Philadelphia Semiconductor Index (SOX), semiconductor book-to-bill ratio, and the semiconductor equipment book-to-bill ratio. The SOX index is

continuously computed while the stock exchange is open. Both book-to-bill ratios are published monthly.

The semiconductor industry is very active. The high-tech companies have consistently shown double digit growth. Most companies have a high beta rating which indicates that their stock price performance is more volatile than lower beta stocks and should be more sensitive to economic conditions. The beta rating for representative stocks is:

Company	Beta
Applied Materials	1.87
Compaq Computers	1.05
IBM	0.58
Digital Equipment	1.55
Intel	1.17
Motorola	0.93
Texas Instruments	1.45

(Source: Standard & Poors Stock Reports)

The importance of the beta characteristic on rule generation was not examined as part of this thesis.

4.3 Data Collection

The data used in this thesis was obtained from Telescan Inc. Telescan has an extensive database and several software products for analysis and evaluation. The database tracks over 7,000 stocks, 2,000 mutual funds, 197 industry groups, 100 market indexes and 40,000 options [35]. The Telescan tool QuoteLink allows downloading historical data as text files in an ASCII-5 format [34]. This tool was used to obtain all data used in this thesis.

4.4 Rule Generator

The rule generator selected for this project is DataLogic/R+ from Reduct and Lobbe Technologies. It can handle tables with up to 2000 attributes and 64,000 objects [28]. The three objectives of the program are to:

1. generate strong, non-redundant classification rules. By non-redundant is meant that it contains the minimum number of condition attributes.
2. permit the analysis of patterns by generating rules that can be inspected and analyzed.
3. permit prediction from new test cases based on the matching of rules to the test cases and producing the most likely outcome.

5 Input Data Preparation

The raw input data follows the preparation steps shown graphically in Figure 7 before running the DataLogic program.

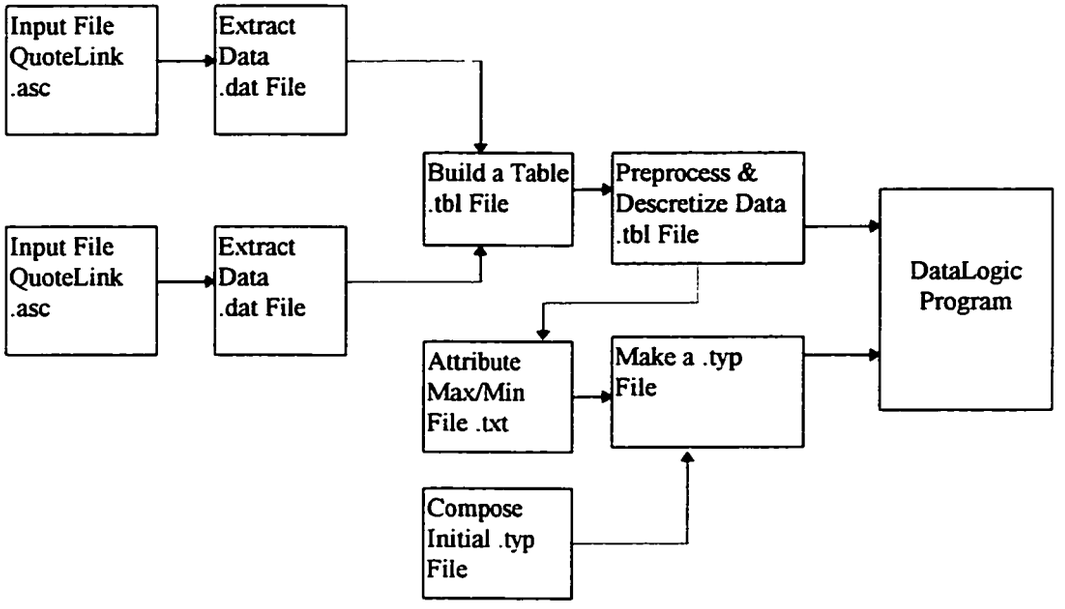


Figure 7. Data Preparation Flow Diagram

Source code for all the computer programs written for data preparation is in Appendix D. Batch programs are also available in Appendix E that combine all the programs. The following is a high level design description of each of the steps.

5.1 Data Extraction

The historic data from QuoteLink has the following comma separated syntax:

symbol, d, date, high, low, close, volume

where symbol = stock or index symbol

d = ASCII character d indicating daily data

date = mm/dd/yr

high = highest price of the day

low = lowest price of the day

close = price of last transaction of the day

volume= number of shares traded that day

The price chosen to be used in the analysis is the closing price of the day. Another choice could have been the median price for the day. This was not investigated. A C-language program called Data.C was written to extract the closing price from the QuoteLink file.

Data.C has as input the file <name>.asc. The output file is named <name>.dat. The program extracts the closing price, computes the percentage change from the closing price of the previous day, and writes the date and percentage change to the output file.

DatMonth is a program very similar to Data, except that the data is averaged for the month before the percentage change is calculated. This program is used with indexes that are available only on a monthly basis.

5.2 Data Preprocessing

Preprocessing of the raw data is necessary to enable finding rules that are predictive. The first step in preprocessing has already been done in the data extraction step which was the calculation of daily percentage change. Calculating the percentage has the effect of normalizing the data. The actual stock price or index level is not important. What is important is the amount of change.

The next step is to run the program Build. This program takes as input the individual data files and converts them into a table that is acceptable to DataLogic. As part of this conversion, the real-numbered percent changes are rounded to the nearest integer value. All data files must have the same number of entries. Each data file becomes a column in the table which is an attribute. The date field of each entry is checked to insure the integrity of the data in the table.

Additional optional preprocessing is done using the programs RufDelay.C, RufAver.C, and RufSum.C. Each of these programs takes as input the table produced by Build and computes a new table with different characteristics. All three programs are flexible and will handle different size tables. The number of rows and columns are input parameters to the programs. In the RufDelay program the original table is reproduced six additional times. Each time it is reproduced the rows or objects are shifted by one row. The attribute name is appended with a number indicating the number of delays. Since the rows

represent a time series, the effect is to create a leading indicator. The new table is seven times wider than the original table. It is shown graphically is Table 6.

	aaa	bbb	ccc	aaa1	bbb1	ccc1		aaa6	bbb6	ccc6
1	1	1	1							
2	2	2	2	1	1	1				
3	3	3	3	2	2	2				
4	4	4	4	3	3	3	→			
5	5	5	5	4	4	4				
6	6	6	6	5	5	5				
7	7	7	7	6	6	6		1	1	1
8	8	8	8	7	7	7		2	2	2

Table 6. Partial RufDelay Table showing how data is delayed with each of the six reproductions of the original table.

The first six rows of the table are deleted because they contain empty cells. When this new table called RufDelay.tbl is run, it will look for rules using data that is from one to six time periods earlier. The choice of delays from one to six time periods with each delay being one time period was arbitrary. Additional tests could be run with more delays and/or with each delay being more than one time period.

The program RufAver also reproduces the original table six more times. In this case, for each reproduction, the value of each row or object is the average of the previous 5, 10,

15, 20, 25, and 30 values respectively, in each column or attribute. It is shown graphically is Table 7.

	aaa	bbb	aaa1	bbb1	aaa2	bbb2
1	1	1				
2	2	2				
3	3	3				
4	4	4				
5	5	5	(1+2+3+4+5)/5	(1+2+3+4+5)/5		
6	6	6	(2+3+4+5+6)/5	(2+3+4+5+6)/5		
7	7	7	(3+4+5+6+7)/5	(3+4+5+6+7)/5		
8	8	8	(4+5+6+7+8)/5	(4+5+6+7+8)/5		
9	9	9	(5+6+7+8+9)/5	(5+6+7+8+9)/5		
10	10	10	(6+7+8+9+10)/5	(6+7+8+9+10)/5	Sum(1:10)/10	Sum(1:10)/10
11	11	11	(7+8+9+10+11)/5	(7+8+9+10+11)/5	Sum(2:11)/10	Sum(2:11)/10
12	12	12	(7+8+9+10+11)/5	(7+8+9+10+11)/5	Sum(3:12)/10	Sum(3:12)/10

Table 7. Partial RufAver Table showing how data is averaged with each of the six reproductions of the original table.

Forming a moving average is a common technique used to filter out noise from data. In the case of the stock market, it may be a way to filter out coincident events. RufAver is

used with daily data. The program mRufAver is used with monthly data where the averages are 1, 2, 3, 4, 5, and 6 month periods. . It is shown graphically is Table 8.

	aaa	bbb	aaa1	bbb1	aaa2	bbb2	aaa6	bbb6
1	1	1	1/1	1/1				
2	2	2	2/1	2/1	1+2/2	1+2/2		
3	3	3	3/1	3/1	2+3/2	2+3/2		
4	4	4	4/1	4/1	3+4/2	3+4/2	→	
5	5	5	5/1	5/1	4+5/2	4+5/2		
6	6	6	6/1	6/1	5+6/2	5+6/2	(1+2+3+4+5+6)/6	(1+2+3+4+5+6)/6
7	7	7	7/1	7/1	6+7/2	6+7/2	(2+3+4+5+6+7)/6	(2+3+4+5+6+7)/6
8	8	8	8/1	8/1	7+8/2	7+8/2	(3+4+5+6+7+8)/6	(3+4+5+6+7+8)/6

Table 8. Partial mRufAver Table showing how data is averaged with each of the six reproductions of the original table.

The RufSum program also duplicates the original table six times. Here for each new duplicate, the field entry is the sum of the previous 1, 2, 3, 4, 5, and 6 rows. This creates a new table to investigate the cumulative effects on the decision attribute. It is shown graphically is Table 9.

	aaa	bbb	aaa1	bbb1	aaa2	bbb2		aaa6	bbb6
1	1	1	1	1					
2	2	2	2	2	1+2	1+2			
3	3	3	3	3	2+3	2+3			
4	4	4	4	4	3+4	3+4	→		
5	5	5	5	5	4+5	4+5			
6	6	6	6	6	5+6	5+6		1+2+3+4+5+6	1+2+3+4+5+6
7	7	7	7	7	6+7	6+7		2+3+4+5+6+7	2+3+4+5+6+7
8	8	8	8	8	7+8	7+8		3+4+5+6+7+8	3+4+5+6+7+8

Table 9. Partial RufSum Table showing how data is summed with each of the six reproductions of the original table.

5.3 Data Discretization

Discretization is the process of converting the continuous range of values for the attributes into discrete values. The raw data for some of the economic indexes used e.g. book-to-bill ratio and SOX semiconductor index, are continuous values. The values must be divided into finite number of values. The initial method selected was to convert the percentage changes to integers by rounding the decimal numbers to the nearest integer. Further reduction of the values has been used in other studies [7]. Another method would

be to simplify the ranges to positive or negative changes. This would indicate direction of the change but not the amount. This may be satisfactory for some users.

The number of values for the attributes impacts the running time of the DataLogic program. The supplier suggests using ten values initially.

5.3.1 Compose .TYP File

The DataLogic program also requires a .typ file in addition to the table file .tbl. The .typ file contains the format data for the table including the column labels, the number of rows or objects in the table, the type of data e.g. integer, real, or string, the field size, and the maximum and minimum values of each column in the table. This file is generated automatically if the table is created within the program by entering the values manually into the spreadsheet form. In our case, the data is much too large to be reentered by hand into the spreadsheet. Therefore the .typ file will be created externally. First the user composes the initial .typ file using any text editor. The file format is obtained from any existing .typ file. Next, the maximum and minimum values for the attributes are calculated by the preprocessing program and contained in the file *.txt. The final step is to run the program Make_Typ which adds the maximum and minimum values into the initial .typ file. This completes the data preparation phase.

6 Experiment Preparation

The scope of the learning experiment was to develop rules using historical data. A six year data range of August 1, 1990 to July 31, 1996 was selected. This provided 1518 cases or objects for the daily experiment and 72 cases for the monthly experiment.

The next step is the preparation for running the experiment. The experiment will be to find rules about the stock price of the company Applied Materials. The rule-generation program DataLogic/R+ will be used. The decision attribute is the stock price of Applied Materials.

6.1 Data Selection

The selection of condition attributes for the decision table was made based on:

1. data that were available on a monthly basis as minimum and ideally available on a daily basis
2. data that were related to the semiconductor equipment industry i.e., part of the food chain.

The justification for requiring daily data was to examine the capability of generating rules in real time. Ideally, if data were available intra-day and strong rules were generated, then rough sets would be a valuable and flexible tool for daily stock traders. Unfortunately, only limited data are available on a daily basis. Other related indexes were found that were only published monthly. The monthly indexes were:

Semiconductor Book-to-Bill Ratio [29]

Semiconductor Equipment Book-to-Bill Ratio [30]

Semiconductor Front-End Equipment Book-to-Bill Ratio [30]

The daily experiment attributes selected were:

Condition Attribute	Type	Symbol
Dow Jones Industrial Average	Index	Dow
Compaq Computer	Stock	CPQ
IBM	Stock	IBM
Digital Equipment	Stock	DEC
Intel	Stock	Intel
Motorola	Stock	Mot
Texas Instruments	Stock	TI
Semiconductor Index	Index	Semi
Electronics Index	Index	Elec
Electronic Equipment Index	Index	Equip
Decision Attribute	Type	Symbol
Applied Materials (daily price)	Stock	Applied

Additionally, a monthly experiment was constructed. The attributes selected were:

Condition Attribute	Type	Symbol
Semiconductor Book-to-Bill	Index	SIABB

SEMI Book-to-Bill	Index	BB
SEMI Book-to-Bill Front End	Index	BBFE
Decision Attribute	Type	Symbol
Applied Materials (monthly average)	Stock	Applied

All three condition attributes are three-month moving averages.

6.2 Data Preparation

All the attribute data for the daily experiments were discretized into integer values. The real-number closing price percentage change was rounded to the nearest integer value. The basic decision table had 1517 rows and 11 columns. Next the three programs for delaying, averaging, and summing were run on this table thus producing three new tables each with 77 columns and 1511, 1487, and 1511 rows respectively

The basic monthly decision table had 4 columns and 72 rows. After running the monthly conversion programs the new table sizes were 28 columns by 60 rows for all three programs. The decision attribute data for the monthly experiment was constructed by averaging the daily closing price over the month.

The decision attribute was discretized into three values which were -1, 0, and 1 using the formula in the following table:

Decision Attribute % Change	Discretized Value
$x < -0.5$	-1
$-0.5 \leq x < 0.5$	0
$x \geq 0.5$	1

Using this discretization would provide strong predictive rules hopefully that provide insight into the direction of the change but no information about the amount of the change. The 0 value was selected to provide a guard band against noise.

7 Analysis of Results

Preliminary experiments were tried with the decision attribute being an integer value representing the percentage change from the previous case. The daily percentage change ranged from -51 to + 17 for the decision attribute. This would provide 68 classifications or concepts to be evaluated. The resulting rules were weak and possessed only a few supporting cases. This led to the choice of allowing only three classifications. The condition attributes remained unchanged as integer values. The rules generated for the daily and monthly experiments will now be listed. The detailed rule reports from the tool are in Appendix B.

7.1 Daily Rules

The three programs were run in the rule generator program for the decision attribute of Applied Materials daily stock price discretized into three categories of falling, no change, and rising. The rules developed for each of the categories are:

Falling Rules

1. [EQUIP1 < 1] & [TI1 < -2] & [-4 <= SEMI1 <= -1] → [APPLIED = -1]

   Stock price will fall if the 5 day average of EQUIP is less than 1% and the 5 day average of TI is less than -2% and the 5 day average of SEMI is between -4% and -1% (source: average program). This rule is supported by 62.2% of the 254 cases.

2. [-20 <= CPQ1 <= -13] → [APPLIED = -1]

   Stock price will fall if yesterday CPQ was between -20% and -13% (source: delay program). This rule is supported by 100% of the 2 cases.

3. [-10 <= EQUIP2 <= -1] → [APPLIED = -1]

   Stock price will fall if the two day sum of EQUIP was between -10% and -1% (source: sum program). This rule is supported by 65.5% of the 554 cases.

No Change Rules

4. [-2 <= TI1 <= 1] & [1 <= DEC1 <= 2] & [MOT2 >= 2] → [APPLIED = 0]

   Stock price will not change if the 5 day average of TI is between -2% and 1% and the 5 day average of DEC is between 1% and 2% and the 10 day average of MOT is greater than or equal to 2% (source: average program). This rule is supported by 66.7% of the 3 cases.

Rising Rules

5. [EQUIP1 < -3 or EQUIP1 > 0] & [TI1 >= 1] → [APPLIED = 1]

   Stock price will rise if the 5 day average of EQUIP is either less than 3% or greater than 0% and the 5 day average of TI is greater than or equal to 1% (source: average program). This rule is supported by 62.5% of the 253 cases.

6. [SEMI1 < -4 or SEMI1 > -1] & [-3 <= EQUIP1 <= 0] & [TI1 <= -3 or TI1 >= 4] → [APPLIED = 1]

   Stock price will rise if the 5 day average of SEMI is either less than -4% or greater than -1% and the 5 day average of EQUIP is between -3% and 0% and the 5 day average of TI is either less than or equal to -3% or greater than or equal to 4% (source: average program). This rule is supported by 100% of the 3 cases.

7. [-4 <= SEMI1 <= -1] & [TI1 >= 1] → [APPLIED = 1]

   Stock price will rise if the 5 day average of SEMI is between -4% and -1% and the 5 day average of TI is greater than or equal to 1% (source: average program). This rule is supported by 77.8% of the 9 cases.

8. [-8 <= IBM1 <= -5] & [MOT5 = 1] → [APPLIED = 1]

   Stock price will rise if yesterday IBM is between -8% and -5% and 5 days ago MOT was equal to 1% (source: delay program). This rule is supported by 100% of 4 cases.

9. [-2 <= IBM6 <= 0] & [-8 <= IBM1 <= -4] & [MOT5 <= -1 or MOT5 >= 2] → [APPLIED = 1]

Stock price will rise if 6 days ago IBM is between -2% and 0% and yesterday IBM was between -8% and -4% and 5 days ago MOT was either less than or equal to -1% or greater than or equal to 2% (source: delay program). This rule is supported by 100% of the 3 cases.

10. [IBM6 < -2 or IBM6 > 0] & [IBM1 <= -7 or IBM1 >= 1] & [MOT5 <= -2 or MOT5 >= 2] → [APPLIED = 1]

    Stock price will rise if 6 days ago IBM was either less than -2% or greater than 0% and yesterday IBM was either less than or equal to -7% or greater than or equal to 1% and 5 days ago MOT was either less than or equal to -2% or greater than or equal to 2% (source: delay program). This rule is supported by 60% of the 180 cases.

11. [EQUIP2 >= 1] → [APPLIED = 1]

    Stock price will rise if the two day sum of EQUIP is greater than or equal to 1% (source: sum program). This rule is supported by 65.9% of the 627 cases.

7.2 Monthly Rules

The rules generated when analyzing the monthly condition attributes Semiconductor Book-to-Bill Index (SIABB), SEMI Equipment Book-to-Bill Index (BB), SEMI Front-End Equipment Book-to-Bill Index (BBFE), and the decision attribute Applied Materials (average monthly stock price) are:

Falling Rules

12. [-5 <= SIABB4 <= 1] & [-25<= SIABB6 <= -4] → [APPLIED = -1]

Stock price will fall if the cumulative four month change in SIABB was between -5% and 1% and the cumulative six month change in SIABB is between -25% and -4%. (source: sum program) This rule is supported by 75.0% of 8 cases.

13. [SIABB6 <= -2] & [-6 <= SIABB5 <= 0] → [APPLIED = -1]

Stock price will fall if the six month average of SIABB is less than or equal to -2% and the five month average of SIABB was between -6% and 0%. (source: average program) This rule is supported by 60.0% of the 15 cases.

No Change Rules

No monthly rules were found in this category.

Rising Rules

14. [2 <= BB4 <= 5] & [BBFE4 < 3 or BBFE4 > 6] → [APPLIED = 1]

Stock price will rise if the average change for the last four months of BB is between 2% and 5% and for the last four months if BBFE is less than 3% or greater than 6% (source: average program). This rule is supported by 71.4% of the 7 cases.

15. [3 <= BBFE4 <= 6] & [SIABB6 >= -2] → [APPLIED = 1]

Stock price will rise the average change for the last four months of BBFE is between 3% and 6% and the change for the last six months of SIABB is greater than -2% (source: average program). This rule is supported by 93.3% of the 15 cases.

16. [2 <= BB4 <= 10] & [1 <= BBFE3 <= 7] & [BB1 <= -4 or BB1 >= 2] → [APPLIED = 1]

Stock price will rise if the change four months ago of BB was between 2% and 10% and for three months ago of BBFE was between 1% and 7% and for one month ago of BB was less than -4% or greater than 2% (source: delay program). This rule is supported by 100% of the 10 cases.

17. [-11 <= BBFE3 <= -5] & [BB1 <= -7 or BB1 >= 2] → [APPLIED = 1]

Stock price will rise if three months ago the BBFE changed between -11% and -5% and last month BB changed either less than -7% or greater than 2% (source: delay program). This rule is supported by 100% of the 5 cases.

18. [1 <= BBFE3 <= 7] & [BB1 >= 2] → [APPLIED = 1]

Stock price will rise if three months ago the BBFE changed between 1% and 7% and last month the BB changed greater than 2% (source: delay program). This rule is supported by 92.9% of the 14 cases.

19. [2 <= BB4 <= 10] & [BB1 >= 2] → [APPLIED = 1]

Stock price will rise if four months ago the BB changed between 2% and 10% and last month the BB changed more than 2% (source: delay program). This rule is supported by 92.9% of the 14 cases.

20. [-6 <= BBFE3 <= 7] & [-8 <= BB1 <= -4] → [APPLIED = 1]

Stock price will rise if three months ago the BBFE changed between -6% and 7% and last month the BB changed between -8% and -4% (source: delay program). This rule is supported by 100% of 2 cases.

21. [BBFE5 >= 6] & [SIABB6 >= -5] → [APPLIED = 1]

Stock price will rise if cumulative sum for 5 months of BBFE is greater than or equal to 6% and the 6 month sum of SIABB is greater than or equal to -5% (source: summing program). This rule is supported by 66.7% of the 6 cases.

22. [BBFE4 < 9 or BBFE4 > 28] & [SIABB6 <= -28 or SIABB6 >= -4] → [APPLIED = 1]

Stock price will rise if the cumulative sum for 4 months of BBFE is either less than 9% or greater than 28% and the 6 month sum of SIABB is either less than or equal to -28% or greater than or equal to -4% (source: summing program). This rule is supported by 62.5% of the 24 cases.

23. [BBFE5 < 6] & [9 <= BBFE4 <= 28] & [-29 <= SIABB6 <= -3] → [APPLIED = 1]

Stock price will rise if the cumulative sum for 5 months of BBFE is less than 6% and the 4 month sum of BBFE is between 9% and 28% and the 6 month sum of SIABB is between -29% and -3% (source: summing program). This rule is supported by 100% of 1 case.

8 Rule Validation

The rules are validated by comparing them against recent test data and determining the percentage of correct predictions. The daily test data used the eight month period from August 1, 1996 to April 9, 1997. The monthly test data used the five month period from August 1, 1996 to December 31, 1996. The period could not be made larger because the SIA index was changed in January, 1997 [33]. The raw data was preprocessed using the same programs to produce the delayed, averaged, and cumulative tables. The tables were reviewed manually to determine the rule compliance. The spreadsheet program Excel by Microsoft was used as an aid to sort the table based on rule attribute values. The Excel spreadsheets are in Appendix C. The following table lists for each rule the number of cases found, the percent of correct cases, the number of cases found during rule generation, and the percent of correct cases during rule generation:

8.1 Daily Validation Experiment

Rule No.	No. Cases	Correct %	Learning Cases	Learning %
1.	0	NA	254	62.2
2.	0	NA	2	100
3.	58	70.7	554	65.5
4.	3	0.0	3	66.7
5.	46	60.8	253	62.5
6.	0	NA	3	100
7.	2	50.0	9	77.8
8.	0	NA	4	100
9.	1	0.0	3	100
10.	19	68.4	180	60.0
11.	72	80.5	627	65.9

The daily validation results can be separated into two groups. The first group, consisting of rules 4 and 7, includes those rules that came from a small number of supporting cases (less than 10). In this group, the error rates were very high. In the second group consisting of rules generated with a large number of supporting cases (rules 3, 5, 10, and 11), the percentage correct was very close to the percentage from the learning experiment. The differences ranged from -1.7% to 14.6%. Rules 1, 2, 6, and 8 had no instances in the validation experiment.

8.2 Monthly Validation Experiment

Rule No.	No. Cases	Correct %	Learning Cases	Learning %
12.	0	NA	8	75.0
13.	0	NA	15	60.0
14.	0	NA	7	71.4
15.	0	NA	15	93.3
16.	0	NA	10	100
17.	1	100	5	100
18.	0	NA	14	92.9
19.	0	NA	14	92.9
20.	1	100	2	100
21.	0	NA	20	95.0
22.	3	66.7	24	62.5
23.	0	NA	1	100

All rules that had instances (17, 20, and 22) agreed very closely with learning experiment. The differences ranged between 0 and 4.2%.

9 Conclusions

The following conclusions were reached:

1. The application of rough set theory to economic and stock market data provides strong predictive rules. This has been shown with the experiments in this thesis for both daily and monthly data.
2. The use of percentage change provides an easy way to normalize the data.
3. The combined use of delay, average, and cumulative preprocessing of data are useful tools for analyzing time series data such as stock market and economic data.
4. The ratio of rising to falling rules derived was 2.3 to 1. This is believed to have been due to the dominant bull market occurring during the six year learning period.
5. When working with continuous or semi-continuous data such as stock market data, rough set theory does not provide a means for selecting or optimizing the discretizing ranges. It was through trail and error that the ranges used in this thesis were chosen.

Bibliography

[1] O. Aasheim, H. Solheim. Rough Sets as a Framework for Data Mining, Project Report, Knowledge Systems Group, The Norwegian University of Science and Technology, Trondheim, Norway, April 30, 1996.

[2] J. Baltzersen. An Attempt to Predict Stock Market Data, Diploma Thesis, Norwegian Institute of Technology, University of Trondheim, Trondheim, Norway, January 8, 1996.

[3] J. Cohen, E. Zinbarg, and A. Zeikel. *Investment Analysis and Portfolio Management*, Richard D. Irwin, Inc., 1973.

[4] P. Cootner. ed., *The Random Character of Stock Market Prices*, M.I.T. Press, 1964.

[5] J. Deogun, V. Raghavan, A. Sarkar, and H. Sever. Data Mining: Trends in Research and Development, *Rough Sets and Data Mining, Analysis of Imprecise Data*, ed. T. Y. Lin, N. Cercone, Kluwer Academic Publishers, 1997.

[6] J. Gersting. *Mathematical Structures for Computer Science*, Second Edition, W. H. Freeman and Company, New York, 1987.

[7] R. Golan. Stock Market Analysis Utilizing Rough Set Theory, Thesis, University of Regina, Saskatchewan, Canada, February 11, 1995.

[8] J. Grzymala-Busse, A. Wang. Modified Algorithms LEM1 and LEM2 for Rule Induction from Data with Missing Attribute Values, *Joint Conference of Information Sciences*, Research Triangle Park, North Carolina, March 1-5, 1997.

[9] J. Grzymala-Busse. LERS - A System for Learning From Examples Based on Rough Sets, *Intelligent Decision Support, Handbook of Applications and Advances of the Rough Set Theory*, ed. R. Slowinski, Kluwer Academic Publishers, 1992.

[10] J. Grzymala-Busse. Managing Uncertainty in Machine Learning From Examples, *Intelligent Information Systems Proceedings of the Workshop*, Wigry, Poland, June 6-10, 1994.

[11] J. Hjulstad. Mining Propositional Default Rules, Diploma Thesis, Knowledge Systems Group, University of Trondheim, Trondheim, Norway, February 22, 1996.

[12] M. Holsheimer, A. Siebes. Data Mining The Search for Knowledge in Databases, Technical Report CS-R9406, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, 1994.

[13] E. Howrey. Are Stock Prices Random or Predictable?, *Business Economics*, Summer, 1965.

[14] Xiaohua Hu, Nick Cercone, Wojciech Ziarko. GRS: A Generalized Rough Sets Model for Data Mining Applications, *Joint Conference of Information Sciences*, Research Triangle Park, North Carolina, March 1-5, 1997.

[15] J. Jelonek, K. Krewiec, R. Slowinski, J. Stefanowski, and J. Szymas. Rough Sets as an Intelligent Front-End for the Neural Network, *Proceedings of the First National Conference on Neural Networks and Their Applications*, Kule, Czestochowa, April 12-15, 1994.

[16] T. Y. Lin. An Overview of Rough Set Theory from the Point of View of Relational Databases, *Bulletin of International Rough Set Society*, vol. 1, no. 1, 1997.

[17] T. Y. Lin. Coping With Imprecise Information - "Fuzzy" Logic, *Downsizing Expo*, Santa Clara, CA, August 3-5, 1993.

[18] T. Y. Lin. Granular Computing on Binary Relations I: Data Mining and Neighborhood Systems, *Rough Sets in Knowledge Discovery*, ed. A. Skowron, L. Polkowski, Springer-Verlag, to appear 1997.

[19] T. Y. Lin. Granular Computing on Binary Relations II: Data Mining and Neighborhood Systems, *Rough Sets in Knowledge Discovery*, ed. A. Skowron, L. Polkowski, Springer-Verlag, to appear 1997.

[20] T. Y. Lin. Topological and Fuzzy Rough Sets, *Intelligent Decision Support, Handbook of Applications and Advances of the Rough Set Theory*, ed. R. Slowinski, Kluwer Academic Publishers, 1992.

[21] T. Y. Lin and Y. Y. Yao. Mining Soft Rules Using Rough Sets and Neighborhoods, Symposium on Modeling, Analysis, and Simulation, *Computational Engineering in Systems Applications '96 IMACS Multiconference*, Lille, France, July 9-12, 1996.

[22] T. Y. Lin and Q. Liu. Rough Approximate Operators - Axiomatic Rough Set Theory, *Rough Sets, Fuzzy Sets, and Knowledge Discovery*, ed. W. Ziarko, Springer-Verlag, 1994.

[23] T. Mollestad and A. Skowron. A Rough Set Framework for Data Mining of Propositional Default Rules, *9th International Symposium on Methodologies for Intelligent Systems*, ISMIS'96, Zokopane, Poland, June 9-13, 1996.

[24] A. Ohrn, J. Komorowski. ROSETTA - A Rough Set Toolkit for Analysis of Data, *Joint Conference of Information Sciences*, Research Triangle Park, North Carolina, March 1-5, 1997.

[25] Z. Pawlak. Rough Sets *Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, 1991.

[26] Z. Pawlak. Remarks made during keynote speech, *Joint Conference of Information Sciences*, Research Triangle Park, North Carolina, March 1-5, 1997.

[27] Z. Pawlak. Rough Sets, *Rough Sets and Data Mining, Analysis of Imprecise Data*, ed. T. Y. Lin, N. Cercone, Kluwer Academic Publishers, 1997.

[28] DataLogic/R and DataLogic/R+ Rough Sets Guide, version 1.3, Reduct and Lobbe, Regina, Saskatchewan, Canada, 1993.

[29] Semiconductor Industry Association, Seasonally Adjusted Book-To-Bill Ratios, San Jose, CA.

[30] Semiconductor Equipment and Materials International, Market Statistics, Historical Book to Bill, Mountain View, CA, 1996.

[31] A. Skowron, C. Rauser. The Discernibility Matrices and Functions in Information Systems, *Intelligent Decision Support, Handbook of Applications and Advances of the Rough Set Theory*, ed. R. Slowinski, Kluwer Academic Publishers, 1992..

[32] A. Szladow, W. Ziarko. Rough Sets: Working with Imperfect Data, *AI Expert*, vol. 8, no. 7, July, 1993.

[33] D. Takahashi. New Chip Indicator Is Off to Rocky Start, *Wall Street Journal*, November 13, 1996.

[34] Telescan QuoteLink Users Manual, Telescan, Inc., Houston, Texas, 1993.

[35] Telescan System 3.0 Operations Manual, Telescan Inc., Houston, Texas, 1992.

[36] I. Velasco, T. Y. Lin, D. Teo. Design Optimization of Rough-Fuzzy Controllers Using a Genetic Algorithm, *Joint Conference of Information Sciences*, Research Triangle Park, North Carolina, March 1-5, 1997.

[37] Y. Y. Yao, S. K. M. Wong, and T. Y. Lin, A Review of Rough Set Models, *Rough Sets and Data Mining, Analysis of Imprecise Data*, ed. T. Y. Lin, N. Cercone, Kluwer Academic Publishers, 1997.

[38] Y. Y. Yao. Two Views of the Theory of Rough Sets in Finite Universes, *International Journal of Approximate Reasoning*, vol 15, no.4, 1996.

[39] W. Ziarko, R. Golan, and D. Edwards. An Application of DataLogic/R Knowledge Discovery Tool to Identify Strong Predictive Rules in Stock Market Data, *AAAI-93 Workshop on Knowledge Discovery in Databases*, Washington, DC, 1993.

[40] W. Ziarko. Variable Precision Rough Set Model, *Journal of Computer and System Sciences*, vol. 46, no. 1, February, 1993.

Appendix A - Sample Decision Tables

RufDelay Monthly Preprocessed Data

SIABB	BB	BBFE	APPLIED	SIABB1	BB1	BBFE1	PPLIED1	SIABB2	BB2	BBFE2	PPLIED2	SIABB3	BB3
0	-2	-2	0	0	-3	-2	-4	-1	3	8	2	-6	1
0	7	3	-1	0	-2	-2	0	0	-3	-2	-4	-1	3
1	4	7	-1	0	7	3	-5	0	-2	-2	0	0	-3
5	-5	-4	0	1	4	7	-12	0	7	3	-5	0	-2
1	2	6	1	5	-5	-4	0	1	4	7	-12	0	7
-2	11	16	1	1	2	6	14	5	-5	-4	0	1	4
-2	10	6	1	-2	11	16	26	1	2	6	14	5	-5
-2	-6	-8	1	-2	10	6	15	-2	11	16	26	1	2
4	-10	-12	-1	-2	-6	-8	4	-2	10	6	15	-2	11
1	-9	-9	1	4	-10	-12	-13	-2	-6	-8	4	-2	10
5	-3	-3	1	1	-9	-9	5	4	-10	-12	-13	-2	-6
-3	2	5	-1	5	-3	-3	1	1	-9	-9	5	4	-10
3	2	5	1	-3	2	5	-3	5	-3	-3	1	1	-9
2	4	5	1	3	2	5	19	-3	2	5	-3	5	-3
4	3	0	1	2	4	5	12	3	2	5	19	-3	2
2	2	4	1	4	3	0	8	2	4	5	12	3	2
-5	5	4	1	2	2	4	20	4	3	0	8	2	4
2	4	5	1	-5	5	4	4	2	2	4	20	4	3
-3	3	-1	1	2	4	5	4	-5	5	4	4	2	2
0	-3	-4	1	-3	3	-1	2	2	4	5	4	-5	5
-3	5	5	-1	0	-3	-4	16	-3	3	-1	2	2	4
-2	2	6	1	-3	5	5	-8	0	-3	-4	16	-3	3
1	2	2	1	-2	2	6	23	-3	5	5	-8	0	-3
3	-2	0	1	1	2	2	8	-2	2	6	23	-3	5
-2	-4	-4	1	3	-2	0	10	1	2	2	8	-2	2
-2	-11	-9	1	-2	-4	-4	19	3	-2	0	10	1	2
-2	1	4	-1	-2	-11	-9	4	-2	-4	-4	19	3	-2
0	2	5	1	-2	1	4	-8	-2	-11	-9	4	-2	-4
2	13	11	1	0	2	5	1	-2	1	4	-8	-2	-11
-1	-2	-7	1	2	13	11	7	0	2	5	1	-2	1
1	-3	-6	1	-1	-2	-7	15	2	13	11	7	0	2
0	-5	-9	1	1	-3	-6	13	-1	-2	-7	15	2	13
0	-2	2	-1	0	-5	-9	3	1	-3	-6	13	-1	-2
0	-3	-3	0	0	-2	2	-10	0	-5	-9	3	1	-3
-3	14	18	-1	0	-3	-3	0	0	-2	2	-10	0	-5
0	3	3	1	-3	14	18	-3	0	-3	-3	0	0	-2
2	1	2	1	0	3	3	6	-3	14	18	-3	0	-3
1	-15	-14	-1	2	1	2	12	0	3	3	6	-3	14
-5	9	12	-1	1	-15	-14	-5	2	1	2	12	0	3

RufDelay Daily Preprocessed Data

DOW	CPO	IBM	DEC	INTEL	MOT	TI	SEM	ELEC	EQUIP	AFFLIED	DOW1	CPO1	IBM1	1-Dec	INTEL1	MOT1	TI1	SEM1
-2	-1	-1	1	-3	-1	-2	-2	-2	3	-1	1	0	0	-1	6	0	-1	3
1	4	2	0	0	2	3	0	-1	0	0	-2	-1	-1	1	-3	-1	-2	-2
0	-4	1	2	-3	-1	0	-1	4	0	-1	1	4	2	0	0	2	3	0
0	-1	1	-1	0	-1	2	0	3	3	1	0	-4	1	2	-3	-1	0	-1
-2	-4	-1	-3	-3	-2	-3	-3	-3	-3	-1	0	-1	1	-1	0	-1	2	0
-1	-3	-2	-1	-5	-2	-3	-4	-3	-3	-1	-2	-4	-1	-3	-3	-2	-3	-3
0	-5	1	-1	-3	-3	0	-2	2	0	-1	-1	-3	-2	-1	-5	-2	-3	-4
-2	-3	-2	-1	-3	-3	-1	-3	-4	-1	1	0	-5	1	-1	-3	-3	0	-2
-2	-3	-1	-3	0	0	-2	-1	1	3	1	-2	-2	-2	-1	-3	-3	-1	-3
-3	1	-2	-3	3	-4	-2	-1	-3	-4	-1	-2	-2	-1	-2	0	0	-2	-1
2	3	4	1	8	5	3	5	5	0	-1	-3	1	-2	-3	3	-4	-2	-1
-2	3	4	1	1	4	2	3	6	3	1	2	3	4	1	8	5	3	5
0	3	-1	3	-3	2	-3	-1	1	-1	-1	3	3	4	1	1	4	2	3
1	0	0	0	-2	-1	3	-1	1	1	0	0	3	-1	3	-2	2	-3	-1
-1	-4	-2	-1	-5	-2	-1	-3	0	-3	-1	1	0	0	0	-3	-1	3	-1
1	-4	0	1	1	2	0	1	-1	2	1	-1	-4	-2	-1	-5	-2	-1	-3
0	3	1	1	0	-1	0	0	-4	-1	1	1	-4	0	1	1	2	0	1
1	-5	0	2	-3	-3	-1	-1	-1	2	1	0	3	1	1	0	-1	0	0
-1	0	0	-3	-4	-11	-4	-3	-1	-3	-1	1	-5	0	2	-3	-3	-1	-1
1	-2	3	1	2	0	0	1	1	2	1	-1	0	0	-3	-4	-11	-4	-3
0	1	0	0	2	0	-2	0	2	1	1	1	-2	3	1	2	0	0	1
0	-4	2	-2	-2	-2	14	1	-2	-2	-1	0	1	0	0	2	0	-2	0
1	2	0	-3	2	2	-2	1	1	0	-1	0	-4	2	-2	-2	-2	14	1
-2	-2	-3	-4	-3	1	-4	-2	0	-1	-1	1	2	0	-3	2	2	-2	1
-1	2	-1	-5	1	3	4	0	-1	-1	-1	-2	-2	-3	-4	-2	1	-4	-2
0	-1	1	0	1	0	2	0	-2	-1	1	-1	2	-1	-5	1	3	4	0
0	-1	2	-2	2	-2	0	0	-1	-1	-1	0	-1	1	0	1	0	2	0
-1	3	1	0	1	-1	-1	1	-1	1	-1	0	-1	2	-2	2	-2	0	0
-2	-5	0	-3	-3	-1	-3	-3	-2	-2	1	-1	3	1	0	1	-1	-1	1
0	-3	-1	-2	0	0	-4	-1	2	0	1	-2	-5	0	-3	-3	-1	-3	-3
-2	-4	-2	-1	-3	-1	-2	-3	-3	0	-1	0	-3	-1	-2	0	0	-4	-1
1	5	1	-1	5	4	5	4	2	1	1	-2	-4	-2	-1	-3	-1	-2	-3
-1	-2	-1	-1	-3	-1	-3	-2	-5	-1	-1	1	5	1	-1	5	4	5	4
-1	-2	-1	0	-2	-1	0	-2	0	-1	-1	-1	-2	-1	-1	-3	-1	-3	-2
1	6	2	2	4	5	5	3	-1	-2	-1	-1	-2	-1	0	-2	-1	0	-2
3	5	2	2	2	5	-2	2	7	3	1	1	6	2	2	4	5	5	3
0	-1	-1	0	1	1	-2	1	5	3	1	3	5	2	2	2	5	-2	2
-1	-4	-1	-1	-4	-4	-1	-3	-2	-2	-1	0	-1	-1	0	1	1	-2	1
1	-3	0	-5	-3	-5	0	-2	2	-3	-1	-1	-4	-1	-1	-4	-4	-1	-3

Appendix B - Rule Reports

Daily - Average

```
          ********   Rule File :RUFAVER.prr   ********

This procedure started at 09:28:37  04-15-1997

System Setup  : Roughness                  0.93
                Rule Precision Threshold  0.55


Decision :: APPLIED       ==> -1,0
====================================
    1 |   |   |[EQUIP1        =1] & [TI1         =2] & [-4<= SEMI1          =-1]


Decision :: APPLIED       ==> -1,0
====================================
    2 |   |   |[-2<= TI1          =1] & [1<= DEC1          =2] & [MOT2          =2]


Decision :: APPLIED       ==> 1
================================
    3 |   |   |[EQUIP1        =-3 or EQUIP1          =0] & [TI1          =1]
      | OR
    4 |   |   |[SEMI1         =-4 or SEMI1          =-1] & [-3<= EQUIP1          =0] &
      |   |   |[TI1         =-3 or TI1         =4]
      | OR
    5 |   |   |[-4<= SEMI1          =-1] & [TI1          =1]


                    ******** Attribute Strength Report ********

Decision Attribute :APPLIED

Decision : APPLIED       ==> -1,0     Coverage ==> 25.32%
============================================================

          Attribute          Max.Loc.Str
          ===========        ============
          EQUIP1               0.16
          TI1                  0.16
          SEMI1                0.15


Decision : APPLIED       ==> -1,0     Coverage ==> 1.12%
============================================================

          Attribute          Max.Loc.Str
          ===========        ============
          TI1                  0.11
          MOT2                 0.11
          DEC1                 0.10


Decision : APPLIED       ==> 1     Coverage ==> 24.38%
=========================================================

          Attribute          Max.Loc.Str
          ===========        ============
```

TI1	0.13
SEMI1	0.12
EQUIP1	0.12

******* Rule Strength Report *******

Rule #	Rule Prob.%	Total Cases	Supporting Cases (#)
1	62.2	254	6,7,8,9,10,17,18,19,20,21,22,31,32,33,34,126, 127,131,132,133,154,155,156,157,158,159,160,161, 169,170,171,172,173,185,186,187,188,189,190,195, 196,197,198,199,215,216,217,218,229,235,236,248, 249,250,253,254,266,267,268,281,298,299,300,301, 302,304,369,372,373,374,375,377,389,390,391,392, 393,400,408,410,425,440,442,444,445,446,447,489, 491,492,519,570,571,601,609,636,637,638,645,646, 647,648,656,657,694,703,710,711,754,755,774,775, 776,777,778,779,780,784,795,796,797,798,799,805, 806,807,808,820,821,822,823,894,895,896,897,898, 899,905,906,908,909,910,911,923,926,927,944,945, 946,947,953,954,955,956,972,973,974,975,976, 1004,1005,1006,1007,1018,1019,1020,1021,1022, 1023,1026,1027,1028,1029,1074,1075,1076,1105, 1108,1150,1152,1190,1191,1210,1213,1225,1226, 1227,1235,1236,1237,1238,1248,1250,1251,1252, 1254,1255,1266,1267,1273,1274,1279,1280,1281, 1282,1283,1294,1295,1303,1304,1305,1311,1312, 1313,1314,1315,1316,1322,1323,1324,1330,1332, 1333,1341,1342,1345,1346,1347,1348,1349,1370, 1371,1372,1381,1382,1383,1385,1398,1399,1439, 1440,1441,1451,1452,1458,1460,1471,1472,1473, 1474,1475,1476,1477,1482
2	66.7	3	97,332,1416
3	62.5	253	25,27,28,36,37,38,39,40,42,43,44,52,53,54,55,56, 57,58,59,70,71,72,88,89,98,99,100,101,103,104, 112,120,136,138,139,148,149,150,177,180,181,201, 202,221,223,276,277,278,324,325,326,327,328,329, 331,339,340,341,358,359,360,401,402,403,431,432, 463,464,465,474,475,476,477,499,505,506,507,510, 527,530,531,545,561,584,585,586,606,623,626,627, 630,666,667,668,669,670,671,672,673,678,679,687, 700,716,741,742,743,744,745,790,793,800,801,802, 803,804,813,833,841,842,843,844,845,846,847,858, 869,870,871,872,886,887,888,901,902,903,914,915, 916,918,933,934,950,958,960,961,965,966,967,968, 969,981,982,983,992,1015,1031,1032,1033,1043, 1044,1045,1046,1047,1077,1093,1094,1095,1096, 1097,1112,1113,1114,1115,1116,1117,1118,1119, 1131,1137,1138,1145,1146,1147,1157,1159,1166, 1167,1168,1169,1175,1179,1180,1181,1182,1183, 1186,1202,1204,1205,1206,1207,1208,1216,1217, 1218,1219,1220,1221,1222,1230,1231,1232,1240, 1242,1243,1244,1245,1246,1247,1259,1260,1261, 1262,1288,1317,1318,1319,1336,1354,1355,1356, 1364,1365,1366,1376,1377,1393,1394,1395,1396, 1407,1408,1409,1415,1416,1417,1418,1419,1420, 1421,1486,1487
4	100.0	3	93,588,907
5	77.8	9	204,257,401,659,785,861,1339,1352,1353

This procedure was completed at 09:31:00 04-15-1997

Daily - Delay

```
          ********   Rule File :RUFDELAY.prr   ********

This procedure started at 09:35:12  04-15-1997

System Setup  : Roughness                  0.93
                Rule Precision Threshold  0.55


Decision :: APPLIED       == -1,0
====================================
    1 :    |    |[-20= CPQ1          =-13]


Decision :: APPLIED       == 1
================================
    2 :    |    |[-8= IBM1          =-5] & [MOT5        =1]
       OR
    3 :    |    |[-2= IBM6          =0] & [-8= IBM1          =-4] & [MOT5        =-1 or
           |    |MOT5        =2]
       OR
    4 :    |    |[IBM6        =-2 or IBM6        =0] & [IBM1        =-7 or IBM1        =1] &
           |    |[MOT5        =-2 or MOT5        =2]


                  ******** Attribute Strength Report  ********

Decision Attribute :APPLIED

Decision : APPLIED       == -1,0      Coverage == 0.31%
=========================================================

          Attribute          Max.Loc.Str
          ===========        =============
          CPQ1                 0.10


Decision : APPLIED       == 1      Coverage == 16.57%
=======================================================

          Attribute          Max.Loc.Str
          ===========        =============
          IBM1                 0.08
          MOT5                 0.08
          IBM6                 0.08



                  *******   Rule  Strength Report  *******

          Rule     Total
Rule #   Prob.%   Cases           Supporting Cases (#)
======   ======   =======  ============================================
     1    100.0         2  180,1407
     2    100.0         4  491,664,1423,1436
     3    100.0         3  323,338,721
     4     60.0       180  4,12,13,18,21,23,28,37,50,54,65,71,74,85,110,
                           111,112,116,117,121,126,133,134,153,154,158,173,
                           195,199,227,230,245,250,264,276,279,297,300,317,
                           340,352,363,368,381,382,395,422,424,425,426,430,
                           437,444,466,474,499,518,523,528,530,565,582,596,
                           614,626,627,628,632,639,642,644,647,651,652,653,
                           656,669,676,680,682,686,696,703,706,707,709,712,
                           739,753,775,807,810,811,813,819,826,829,834,835,
```

```
                              856,865,876,885,886,891,896,906,910,912,915,923,
                              925,929,934,937,939,953,959,960,971,972,999,
                              1033,1052,1055,1057,1059,1066,1079,1080,1101,
                              1109,1160,1169,1178,1189,1192,1226,1236,1241,
                              1243,1245,1253,1256,1271,1281,1293,1299,1305,
                              1310,1312,1317,1321,1330,1342,1357,1361,1369,
                              1371,1375,1378,1380,1381,1383,1389,1394,1397,
                              1401,1404,1409,1413,1424,1438,1442,1447,1457,
                              1461,1486,1491,1501

This procedure was completed at 10:09:44  04-15-1997
```

Daily - Sum

```
********   Rule File :RUFSUM.prr   ********

This procedure started at 14:15:41  05-07-1997

System Setup  : Roughness                  0.93
                Rule Precision Threshold  0.55


Decision :: APPLIED       ==·  ·-1,0·
====================================
    1  |    |    |[-10·= EQUIP2          ·=-1]
       | OR
    2  |    |    |[EQUIP2          ·=-11]


Decision :: APPLIED       ==·  1
================================
    3  |    |    |[-12·= EQUIP2          ·=-9]
       | OR
    4  |    |    |[EQUIP2          ·=1]


                  ******** Attribute Strength Report  ********

Decision Attribute :APPLIED

Decision : APPLIED       ==·  ·-1,0·     Coverage ==·  57.14%
============================================================

          Attribute          Max.Loc.Str
          ===========        =============
          EQUIP2               0.35


Decision : APPLIED       ==·  1     Coverage ==·  59.65%
=========================================================

          Attribute          Max.Loc.Str
          ===========        =============
          EQUIP2               0.33



                  *******   Rule  Strength Report  *******

         Rule    Total
Rule #   Prob.%  Cases          Supporting Cases (#)
```

1	65.5	554	2,6,7,8,9,10,11,15,16,19,20,23,24,25,26,27,29, 30,34,35,39,40,41,42,53,54,55,56,57,63,69,70,82, 83,86,87,89,90,97,98,101,102,103,104,105,106, 108,109,114,126,127,130,131,132,145,146,147,148, 151,152,153,154,155,161,165,166,167,168,175,176, 178,179,181,182,187,188,191,192,193,196,197,206, 207,209,210,211,212,215,216,217,218,220,221,227, 228,230,231,234,235,236,240,241,242,248,249,252, 255,256,259,260,265,269,270,273,274,275,280,281, 283,284,285,286,287,290,291,292,293,295,296,305, 306,314,318,322,323,324,325,332,334,335,336,337, 338,344,345,346,359,360,365,366,369,371,372,373, 375,385,386,389,390,393,397,398,401,402,410,411, 414,415,418,419,420,421,428,429,430,433,434,435, 436,440,442,443,444,447,448,457,458,459,464,465, 469,470,472,473,476,480,481,482,483,490,491,494, 495,502,504,505,506,507,514,515,516,517,525,526, 532,533,536,537,539,540,541,543,544,545,550,563, 564,576,586,587,590,593,595,596,599,600,601,611, 612,613,614,619,624,625,634,637,638,640,658,661, 662,668,669,670,671,673,677,678,683,684,685,704, 705,714,715,716,726,727,734,735,742,743,744,745, 767,770,771,772,778,779,783,788,791,792,794,795, 798,799,800,807,808,809,810,819,820,821,829,830, 831,832,833,844,848,849,861,876,877,884,885,888, 889,894,897,898,901,902,906,907,916,917,918,919, 920,921,922,923,927,930,931,933,934,935,944,945, 949,952,953,957,960,967,968,969,970,976,977,980, 994,995,996,997,998,1001,1002,1028,1029,1034, 1041,1042,1045,1050,1051,1052,1057,1058,1059, 1060,1063,1065,1070,1073,1074,1077,1078,1082, 1085,1086,1090,1091,1092,1094,1095,1096,1097, 1102,1103,1104,1107,1112,1113,1123,1124,1125, 1126,1129,1131,1132,1145,1146,1150,1156,1157, 1172,1173,1174,1176,1185,1186,1187,1201,1208, 1212,1213,1214,1215,1220,1224,1233,1234,1235, 1248,1249,1250,1256,1257,1258,1259,1260,1271, 1272,1273,1276,1277,1278,1279,1288,1289,1290, 1291,1294,1295,1296,1297,1298,1301,1302,1303, 1304,1306,1307,1311,1315,1316,1318,1319,1323, 1327,1328,1331,1332,1333,1335,1336,1337,1338, 1339,1344,1345,1346,1347,1348,1351,1352,1353, 1354,1355,1361,1362,1363,1365,1366,1367,1369, 1370,1371,1373,1374,1382,1383,1390,1391,1394, 1395,1398,1402,1403,1404,1405,1406,1407,1408, 1409,1410,1411,1414,1415,1419,1420,1422,1423, 1425,1426,1434,1435,1445,1446,1451,1452,1453, 1455,1458,1459,1461,1462,1463,1466,1467,1471, 1472,1473,1474,1475,1476,1479,1480,1481,1482, 1483,1484,1485,1486,1487,1492,1493,1494,1496, 1497,1498,1499,1500,1503,1504,1505,1506,1510
2	100.0	1	43
3	100.0	1	44
4	65.9	627	1,4,12,13,18,21,36,37,38,46,47,49,50,51,52,59, 60,61,62,65,66,67,72,74,75,76,77,79,80,81,85,88, 92,93,94,95,99,100,107,110,111,112,113,115,116, 117,118,119,120,121,123,124,125,128,129,133,136, 137,138,139,140,141,143,144,157,158,159,162,163, 169,170,172,173,174,185,186,189,194,195,199,200, 201,202,203,204,205,208,213,214,219,223,225,226, 229,232,237,238,243,244,245,246,247,251,261,262, 263,264,266,267,276,277,289,294,297,298,299,300, 301,303,304,310,311,312,316,317,319,320,327,328, 330,333,341,342,347,348,349,350,351,352,357,361, 362,363,364,367,368,370,377,378,382,383,384,387, 388,395,400,404,405,406,407,422,423,424,425,426, 427,432,438,445,450,451,452,453,454,455,456,467, 468,474,478,484,485,487,488,489,492,497,498,499, 500,508,509,511,519,520,523,524,528,529,530,531, 534,535,547,548,551,552,553,554,559,560,562,565,

```
566,568,569,573,574,575,578,579,580,581,582,583,
597,598,605,606,608,609,610,622,623,626,627,628,
630,631,632,635,642,643,644,647,648,650,651,652,
659,660,663,664,666,675,680,681,687,688,689,690,
691,692,693,694,695,696,697,698,699,700,702,703,
706,707,708,711,719,721,722,724,725,729,730,738,
739,740,747,748,749,750,751,752,757,758,759,763,
764,765,766,768,773,774,775,781,782,784,785,789,
790,793,796,797,802,803,804,811,812,813,814,815,
816,817,822,823,824,825,826,827,834,835,836,837,
838,839,851,854,855,856,857,858,859,862,863,865,
866,867,868,869,870,871,874,875,879,880,881,882,
886,887,890,891,892,893,895,896,900,903,904,908,
909,910,911,912,913,924,925,926,929,936,937,938,
939,940,942,948,954,955,958,962,966,971,973,974,
979,981,982,983,984,988,989,990,991,992,993,999,
1003,1004,1005,1006,1011,1012,1013,1016,1017,
1021,1022,1024,1026,1027,1032,1035,1036,1037,
1038,1039,1048,1054,1055,1061,1062,1067,1068,
1069,1075,1076,1080,1087,1088,1089,1093,1100,
1101,1106,1108,1109,1110,1111,1116,1117,1118,
1120,1121,1122,1127,1128,1134,1135,1136,1137,
1139,1140,1141,1142,1143,1144,1148,1149,1152,
1153,1154,1155,1158,1159,1160,1161,1162,1164,
1165,1166,1167,1168,1169,1170,1178,1179,1180,
1183,1188,1189,1190,1191,1192,1196,1197,1200,
1202,1203,1204,1205,1206,1207,1209,1210,1211,
1216,1217,1218,1223,1225,1226,1227,1228,1229,
1230,1231,1232,1236,1237,1238,1239,1240,1241,
1242,1244,1245,1246,1251,1252,1253,1254,1255,
1261,1262,1263,1264,1265,1266,1267,1268,1269,
1270,1274,1280,1281,1282,1283,1286,1287,1292,
1293,1299,1300,1305,1308,1309,1310,1312,1313,
1317,1320,1321,1322,1324,1325,1326,1329,1330,
1334,1341,1342,1343,1350,1357,1359,1360,1364,
1375,1376,1377,1378,1379,1380,1381,1384,1385,
1386,1387,1388,1389,1399,1400,1401,1412,1416,
1417,1418,1421,1424,1427,1429,1430,1431,1432,
1436,1437,1438,1439,1440,1441,1442,1443,1444,
1448,1449,1456,1457,1469,1470,1477,1478,1489,
1490,1501,1502,1507,1508,1509

This procedure was completed at 15:14:31  05-07-1997
```

Month - Average

```
********   Rule File :MRUFAVER.prr   ********

This procedure started at 15:36:59  05-07-1997

System Setup  : Roughness                  0.43
                Rule Precision Threshold   0.55


Decision :: AMAT      ==> -1,0>
================================
    1 |    |    |[SIABB6      >-2] & [-6>= SIABB5       >=0]


Decision :: AMAT      ==> 1
============================
    2 |    |    |[2>= BB4       >=5] & [BBFE4      >3 or BBFE4       >6]
      | OR
    3 |    |    |[3>= BBFE4       >=6] & [SIABB6       >=-2]
```

```
                    ******** Attribute Strength Report  ********

Decision Attribute :AMAT

Decision : AMAT      ==· ·-1,0·      Coverage ==· 47.37%
=========================================================

          Attribute          Max.Loc.Str
          ===========        =============
          SIABB6               0.31
          SIABB5               0.30


Decision : AMAT      ==· 1      Coverage ==· 52.78%
=====================================================

          Attribute          Max.Loc.Str
          ===========        =============
          BB4                  0.33
          BBFE4                0.32
          SIABB6               0.31



                    *******   Rule  Strength Report  *******


          Rule     Total
Rule #   Prob.%    Cases            Supporting Cases (#)
======   ======   =======  ================================================
     1     60.0        15  1,3,21,22,23,24,26,27,30,39,52,53,54,59,60
     2     71.4         7  20,21,22,23,31,41,44
     3     93.3        15  5,6,7,8,14,15,16,17,18,19,29,36,37,42,43

This procedure was completed at 15:37:36  05-07-1997
```

Month - Delay

```
             ********   Rule File :MRUFDELY.prr   ********

This procedure started at 14:12:10  04-14-1997

System Setup  : Roughness                0.93
                Rule Precision Thresnold 0.55


Decision :: AMAT    ==· ·-1,0·
==============================
    1 |    |    |[BBFE    ·=-13 or BBFE    ·=1] & [BBFE3    · =-10 or BBFE3    ·=10]
      | OR
    2 |    |    |[-3·= BBFE     ·=0] & [BBFE3    · =-10 or BBFE3    ·=8]


Decision :: AMAT    ==· 1
=========================
    3 |    |    |[2·= BB4     ·=10] & [1·= BBFE3     ·=7] & [BB1    ·=-4 or BB1    ·=2]
      | OR
    4 |    |    |[-11·= BBFE3     ·=-5] & [BB1    ·=-7 or BB1    ·=2]
      | OR
    5 |    |    |[1·= BBFE3     ·=7] & [BB1    ·=-]
```

```
      OR
    6 |     |    [-2- = BB4     -=10] & [BB1     -=-]
      OR
    7 |     |    [-6- = BBFE3     -=7] & [-8- = BB1     -=-4]


                    ******** Attribute Strength Report  ********

Decision Attribute :AMAT

Decision : AMAT    ==- --1,0-      Coverage ==- 42.11%
=================================================

          Attribute          Max.Log.Str
          ===========        ============
          BBFE                 0.41
          BBFE3                0.40


Decision : AMAT    ==- 1      Coverage ==- 72.22%
=============================================

          Attribute          Max.Log.Str
          ===========        ============
          BBFE3                0.36
          BB1                  0.35
          BB4                  0.33



                    *******   Rule  Strength Report  *******

          Rule     Total
Rule #   Prob.%    Cases          Supporting Cases (#)
======   ======   =======  ============================================
     1    100.0         6  12,38,42,51,56,60
     2     66.7         3  41,49,59
     3    100.0        10  6,16,17,19,20,26,40,43,45,55
     4    100.0         5  11,13,29,47,48
     5     92.9        14  6,8,15,16,17,19,20,24,30,36,40,43,54,55
     6     92.9        14  3,6,7,16,17,18,19,20,22,23,40,43,44,55
     7    100.0         2  5,26

This procedure was completed at 14:10:17  04-14-1997
```

Month - Sum

```
             ********   Rule File :MRUFSUM.prr   ********

This procedure started at 15:23:05  05-07-1997

System Setup  : Roughness                 0.93
                Rule Precision Threshold  0.55


Decision :: AMAT    ==- --1,0>
================================
    1 |     |    [[-5- = SIABB4     -=1] & [-25- = SIABB6     -=-4]
      | OR
    2 |     |    [[-25- = SIABB6     -=-1]
```

```
Decision :: AMAT    == · 1
=========================
    3 :    :    :[BBFE5    ·=6] & [SIABB6    ·=-5]
      . OR
    4 :    :    :[BBFE4    ·9 or BBFE4    ·28] & [SIABB6    ·=-28 or SIABB6    ·=-4]
      . OR
    5 :    :    :[BBFE5    ·6] & [9· = BBFE4    ·=28] & [-29·= SIABB6    ·=-3]


                    ******** Attribute Strength Report  ********

Decision Attribute :AMAT

Decision : AMAT    == · ·-1,0·      Coverage == · 36.84%
=======================================================

          Attribute           Max.Loc.Str
          ===========         =============
          SIABB4                0.36
          SIABB6                0.36


Decision : AMAT    == · 1      Coverage == · 86.11%
=================================================

          Attribute           Max.Loc.Str
          ===========         =============
          SIABB6                0.35
          BBFE5                 0.35
          BBFE4                 0.35



                    *******    Rule  Strength Report  *******


          Rule     Total
Rule #   Prob.%    Cases              Supporting Cases (#)
======   ======   =======   ============================================
    1      75.0         8   3,21,26,27,30,39,53,59
    2     100.0         1   54
    3      95.0        20   5,6,7,8,9,15,16,17,18,19,20,24,25,29,31,36,37,
                            40,43,44
    4      62.5        24   4,9,10,11,12,13,20,28,31,32,33,34,38,40,41,44,
                            45,46,47,48,49,50,51,57
    5     100.0         1   30

This procedure was completed at 15:28:26  05-07-1997
```

Appendix C - Validation Decision Tables

Daily - Average

APPLIED	DEC1	TI1	SEMI1	EQUIP1	MOT2
-1	-2	-2	-1	-2	0
-1	-1	-2	-1	-1	0
0	-1	-2	-1	-1	-1
1	-1	-2	-1	-1	0
-1	-3	-1	-1	-1	0
-1	-2	-1	-1	-1	0
-1	-2	-1	-1	-1	0
-1	-2	-1	-1	-1	0
-1	0	-1	-1	-1	-1
-1	-3	-1	-1	0	0
-1	-2	-1	-1	0	0
-1	2	-1	-1	1	2
0	-2	-1	-1	-1	-1
0	-2	-1	-1	0	0
1	0	-1	-1	-1	0
1	-1	-1	-1	0	-1
-1	-1	0	-1	-1	-1
-1	-1	0	-1	-1	0
-1	-2	0	-1	0	0
-1	-1	0	-1	0	-1
-1	-1	0	-1	0	0
1	-1	0	-1	0	0
0	3	1	-1	1	2
1	-1	1	-1	0	0
1	0	-2	0	1	1
-1	-1	-1	0	-1	0
-1	0	-1	0	-1	-1
-1	0	-1	0	-1	-1
-1	0	-1	0	-1	0
-1	-2	-1	0	0	-1
-1	-2	-1	0	0	0
-1	-1	-1	0	0	-1
-1	0	-1	0	0	0
-1	0	-1	0	0	0
0	-1	-1	0	0	0
0	0	-1	0	0	-1
1	-2	-1	0	-1	1
1	-2	-1	0	-1	1
1	-1	-1	0	-1	0

Daily - Delay

APPLIED	CPQ1	IBM1	MOT5	IBM6
1	0	4	-2	-6
1	1	0	-6	-4
1	1	1	-2	-4
-1	0	-1	3	-3
1	-5	0	-1	-3
1	0	0	0	-3
1	0	0	2	-3
1	2	0	3	-3
1	0	1	-1	-3
1	-4	1	1	-3
1	2	3	1	-3
-1	-6	-3	0	-2
1	-3	-3	-1	-2
-1	-2	-1	-2	-2
-1	-1	-1	-1	-2
0	-4	-1	-1	-2
0	-2	-1	2	-2
1	3	-1	0	-2
1	-3	-1	2	-2
1	-2	-1	2	-2
1	-3	0	0	-2
-1	0	1	2	-2
0	2	1	1	-2
0	6	-6	2	-1
-1	-6	-4	1	-1
-1	-3	-2	-1	-1
-1	-1	-2	0	-1
1	-1	-2	-2	-1
1	-3	-2	4	-1
-1	-4	-1	-2	-1
-1	1	-1	0	-1
-1	0	-1	2	-1
0	4	-1	-1	-1
0	1	-1	0	-1
1	0	-1	0	-1
-1	1	0	-5	-1
-1	1	0	-2	-1
-1	0	0	1	-1
0	1	0	-2	-1

Daily - Sum

APPLIED	EQUIP2
-1	-5
-1	-5
-1	-5
-1	-4
-1	-4
-1	-4
-1	-4
-1	-4
-1	-4
-1	-4
-1	-3
-1	-3
-1	-3
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-2
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1
-1	-1

Month - Sum

APPLIED	SIABB4	BBFE4	BBFE5	SIABB6
-1	-5	6	-5	-4
-1	-3	5	9	-7
-1	-3	-10	-8	-4
-1	-3	16	16	-4
-1	-2	3	20	-5
-1	1	-42	-34	-20
-1	1	6	14	-6
1	-4	9	8	-4
1	-1	12	3	-4
1	0	-12	-6	-4

Month - Delay

BBFE	APPLIED	BB1	BBFE3	BB4
-6	-1	-13	-16	-13
-4	1	-7	-1	-13
10	1	-1	-15	0
13	1	11	-6	-13
9	1	15	-4	-7

Month - Average

APPLIED	BB4	BBFE4	SIABB5	SIABB6
-1	3	5	-7	-6
-1	6	7	-4	-4
-1	-10	-11	-1	-3
0	-9	-8	-5	-5
1	-3	-1	-6	-5
1	6	8	-5	-4

Appendix D - Computer Programs

Data.C

```
/*****************************************************************************
*  PROJECT:          CS 298
*  FILE:             data.c
*  PURPOSE:          Reads a *name*.asc file, selects the closing price,
*                    calculates the percent change from day to day,
*                    and writes the percent change to a *name*.dat file.
*                    *name* is input from the command line. The .asc file
*                    is assumes to be in ASCII 5 format.
*  VERSION:          1.0
*  LANGUAGE:         Borland C++ 4.02
*  TARGET:           IBM PC
*  PROGRAMMER:       Joe Tremba
*  UPDATE HISTORY:   7/31/96 original release
*                    8/14/96 modified to use floating point numbers
*                    8/16/96 added Percent function
*                    9/2/96  added date to .dat file
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ROWMAX 2000
#define COLMAX 1
#define LENMAX 11

/********** Function Prototypes **********************************/
void Initialize(void);
int ReadEntry(FILE *fp);
int WriteEntry(FILE *fp, int rows);
int Percent(int);
void pause(void);

float a[ROWMAX][COLMAX];
char date[ROWMAX][LENMAX];

/*============================================================================
Initialize - puts the number 2000 in all array entries.
input: none
output: none
functions called: none
============================================================================*/
void Initialize()
{
   int i,j;

   for(i = 0; i < ROWMAX; i++)
   {
      for(j = 0; j < COLMAX; j++)
      {
         a[i][j] = 2000.0; /* set array values to 2000 */
      }
      for( j = 0; j < LENMAX; j++)
      {
         date[i][j] = NULL;    /* set date array to NULL */
      }
   }
}

/*============================================================================
ReadEntry - reads date and entries from file.
```

```
Input: none
Output: number of rows read
Functions called: none
*****************************************************************************/
int ReadEntry(FILE *fp)
{  int  ch = 0,
   x = 0,
   y = 0,
   i = 0,
   j = 0,
   k = 0;
   char buffer[LENMAX + 1];
   do
   {
      x = 0;
      while ( ch != 'd' && x < 20)
      {
         ch = fgetc(fp);       /* discard chars until reach 'd' */
         x++;
      }
      if ( x == 20)
      {
         printf("error: x = 20 limit exceeded \n");
         printf("row = %i,  %c\n", i,ch);
         break;
      }
      else if (ch != 'd')
      {
         printf("error - start of file is not 'd'\n");

      }
      x = 0;
      y = 0;
      ch = fgetc(fp);
      if( ch == ',' )       /* put date in array */
      {
         do
         {
            ch = fgetc(fp);
            if (ch != ',')
               date[i][x] = (char) ch;
            x++;
         }
         while( ch != ',' && x < 10 );
      }
      x = 0;
      while ( x < 2 && y < 40)             /* advance to 'close' */
      {
         ch = fgetc(fp);
         if (ch == ',')
            x++;
         y++;
      }
      if( y == 40)
      {
         printf("error: y = 40 limit exceeded \n");

      }
      k = 0;                    /* reset index */
      do
      {
         ch = fgetc(fp);
         if(ch != ',')            /* put price in array */
         {
            buffer[k] = (char)ch;
            k++;
         }
      }
      while( ch != ',' && k < LENMAX );
      buffer[k] = '\0';
      a[i][j] = atof(&buffer[0]);
```

```
      if(a[i][j] > 10000.)
      {
         printf("error: float > 10000 \n" );

      }
      i++;                          /* increment row */
   }
   while (!feof(fp) && i < ROWMAX && j < COLMAX && k < LENMAX );
   return i;
}

/*============================================================================
Percent - computes the percent change from present row and the previous row.
          The zero value in array is no longer valid.
Input: number of rows
Output: 0 if successful, 1 otherwise
Functions called: none
============================================================================*/
int Percent(int row)
{
   int i, j;

   for( i = row -1; i > 0; i--)
   {
      for( j = 0; j < COLMAX; j++)
      {
         if(a[i][j] == 2000.0)
         {
            printf("error: array contains 2000.0 \n");

         }
         a[i][j] = (a[i][j] - a[i-1][j]) * 100.0 / a[i-1][j];
      }
   }
   return 0;
}

/*============================================================================
WriteEntry - writes entries to file, except zero entry.
Input: File pointer, number of rows to write
Output: 0 if success, -1 otherwise
Functions called: none
============================================================================*/
int WriteEntry(FILE *fp, int rows)
{
   int i,j;

   for(i = 1; i < rows; i++) /* skip zer. row */
   {
      for(j = 0; j < COLMAX; j++)
      {
         fprintf(fp, "%s %.3f", &date [i][j], a[i][j]);
      }
      fputc( '\n', fp );               /* add CR at end of line */
   }
   fputc(EOF, fp);
   return 0;
}

/*============================================================================
pause - stops the program until a keyboard character is entered.
Input: none
Output: none
Functions called: none
============================================================================*/
void pause ()
{
   getchar();
}

/*----------------------------------------------------------------------------*/
```

```
void main(int argc, char **argv)
{
   FILE *source, *destin;
   int rows;
   char buffer[14];

   if (argc != 2)
   {  printf("Usage: data <filename>");
      argv[1] = "dummy";
   }
   Initialize();
   argv++;
   strcpy( buffer,*argv );
   strcat( buffer,".asc");
   printf("source is %s \n", buffer);
   source = fopen( buffer,"r");
   if (source == NULL)
   {  printf("%s File not found\n", buffer);
      exit(0);
   }
   strcpy( buffer,*argv );
   strcat( buffer,".dat");
   printf("destin is %s \n", buffer);
   destin = fopen( buffer,"w");
   if (destin == NULL)
   {  printf("%s Unable to open \n", buffer);
      exit(0);
   }
   rows = ReadEntry(source);        /* read input file */
   Percent(rows);                   /* convert data to % change */
   WriteEntry(destin, rows);                  /* write output file */
   fcloseall();

}
```

Datmonth.C

```
/*****************************************************************************
*  PROJECT:          CS 298
*  FILE:             datmonth.c
*  PURPOSE:          Reads a <name>.asc file, selects the closing price,
*                    calculates the daily change in percent, and
*                    averages the price for the month, and writes the
*                    closing monthly percent change to a <name>m.dat file.
*                    <name> is input from the command line. The .asc file
*                    is assumes to be in ASCII 5 format.
*  VERSION:          1.0
*  LANGUAGE:         Borland C++ 4.02
*  TARGET:           IBM PC
*  PROGRAMMER:       Joe Tremba
*  UPDATE HISTORY:   11/30/96 original release
*
*
*
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ROWMAX 2000
#define COLMAX 1
#define LENMAX 11

/*********** Function Prototypes **********************************/
void Initialize(void);
int ReadEntry(FILE *fp);
int WriteEntry(FILE *fp, int rows);
```

```
int Percent(int);
void pause(void);

float a[ROWMAX][COLMAX];
char date[ROWMAX][LENMAX];

/*============================================================================
Initialize - puts the number 2000 in all array entries.
Input: none
Output: none
Functions called: none
============================================================================*/
void Initialize()
{
   int i,j;

   for(i = 0; i < ROWMAX; i++)
   {
      for(j = 0; j < COLMAX; j++)
      {
         a[i][j] = 2000.0; /* set array values to 2000 */
      }
      for( j = 0; j < LENMAX; j++)
      {
         date[i][j] = NULL;   /* set date array to NULL */
      }
   }
}

/*============================================================================
ReadEntry - reads date and entries from file and computes monthly average.
Input: none
Output: number of monthly rows read
Functions called: none
============================================================================*/
int ReadEntry(FILE *fp)
{  int  ch = 0,
   x = 0,
   y = 0,
   i = 0,
   j = 0,
   k = 0,
   m = 0,                    /* index of month array */
   month = 0,
   temp_month = 0;
   char buffer[LENMAX + 1],
         mth[4],             /*buffer to store month */
         year[4],            /* buffer to store year */
         datebuffer[10];     /* buffer to hold date */
   float temp = 0.0,
         sum = 0.0,
         days = 1.0;
   do
   {
      x = 0;
      while ( ch != 'd' && x < 20)
      {
         ch = fgetc(fp);      /* discard chars until reach 'd' */
         x++;
      }
      if ( x == 20)
      {
         printf("error: x = 20 limit exceeded \n");
         printf("row = %i,  %c\n", i,ch);
         temp_month = 0;

      }
      else if (ch != 'd')
      {
         printf("error - start of file is not 'd'\n");
```

```
}
x = 0;
y = 0;
ch = fgetc(fp);
if( ch == ',' )        /* put date in array */
{
   do
   {
      ch = fgetc(fp);
      if (ch != ',')
         datebuffer[x] = (char) ch;
      x++;
   }
   while( ch != ',' && x < 10 );
   datebuffer[x] =  '\0';          /* terminate string*/
   temp_month = atoi(&datebuffer[0]);      /* determine month */
}
if( temp_month != month && month != 0) /* has month changed? */
{
   sum = sum/days;                     /* average the month */
   a[m][j] = sum;                      /* put average in array */
   strcpy(&date[m][0], &mth[0]);       /* put month in date array */
   date[m][2] = '/';
   date[m][3] = '\0';
   strcat(&date[m][0], &year[0]);      /* remove day from date */
   year[0] = datebuffer[6];            /* determine new year */
   year[1] = datebuffer[7];
   year[2] = '\0';
   month = temp_month;                 /* start new month */
   mth[0] = datebuffer[0];
   mth[1] = datebuffer[1];
   mth[2] = '\0';
   days = 1.0;                         /* reset days counter */
   sum = 0.0;                          /* reset sum */
   m++;                                /* advance array */
}
if( month == 0)                        /* set date to start loop */
{
   month = temp_month;                 /* start new month */
   mth[0] = datebuffer[0];
   mth[1] = datebuffer[1];
   mth[2] = '\0';
   year[0] = datebuffer[6];            /* determine new year */
   year[1] = datebuffer[7];
   year[2] = '\0';
}
x = 0;
while ( x < 2 && y < 40)               /* advance to 'close' */
{
   ch = fgetc(fp);
   if (ch == ',')
      x++;
   y++;
}
if( y == 40)
{
   printf("error: y = 40 limit exceeded \n");

   break;
}
k = 0;                    /* reset index */
do
{
   ch = fgetc(fp);
   if(ch != ',')             /* put price in array */
   {
      buffer[k] = (char)ch;
      k++;
   }
}
while( ch != ',' && k < LENMAX );
```

```
      buffer[k] = '\0';
      temp = atof(&buffer[0]);
      sum += temp;                    /* add day price to month sum */
      days++;
      if(temp > 10000.)
      {
         printf("error: float > 10000 \n" );

      }
      i++;                            /* increment row */
   }
   while (!feof(fp) && i < ROWMAX && j < COLMAX && k < LENMAX );
   return m;
}

/*============================================================================
Percent - computes the percent change from present row and the previous row.
          The zero value in array is no longer valid.
Input: number of rows
Output: 0 if successful, 1 otherwise
Functions called: none
============================================================================*/
int Percent(int row)
{
   int i, j;

   for( i = row -1; i > 0; i--)
   {
      for( j = 0; j < COLMAX; j++)
      {
         if(a[i][j] == 2000.0)
         {
            printf("error: array contains 2000.0 \n");

         }
         a[i][j] = (a[i][j] - a[i-1][j]) * 100.0 / a[i-1][j];
      }
   }
   return 0;
}

/*============================================================================
WriteEntry - writes entries to file, except zero entry.
Input: File pointer, number of rows to write
Output: 0 if success, -1 otherwise
Functions called: none
============================================================================*/
int WriteEntry(FILE *fp, int rows)
{
   int i,j;

   for(i = 1; i < rows; i++) /* skip zero row */
   {
      for(j = 0; j < COLMAX; j++)
      {
         fprintf(fp, "%s %.3f", &date [i][j], a[i][j]);
      }
      fputc( '\n', fp );              /* add CR at end of line */
   }
   fputc(EOF, fp);
   return 0;
}

/*============================================================================
pause - stops the program until a keyboard character is entered.
Input: none
Output: none
Functions called: none
============================================================================*/
void pause ()
{
```

```
   getchar();
}

/*-------------------------------------------------------------------------*/
void main(int argc, char **argv)
{
   FILE *source, *destin;
   int i, j,rows;
   char buffer[14];

   if (argc != 2)
   {  printf("Usage: data <filename>");
      argv[1] = "dummy";
   }
   Initialize();
   argv++;
   strcpy( buffer,*argv );
   strcat( buffer,".asc");
   printf("source is %s \n", buffer);
   source = fopen( buffer,"r");
   if (source == NULL)
   {  printf("%s File not found\n", buffer);
      exit(0);
   }
   strcpy( buffer,*argv );
   strcat( buffer,"m.dat");
   printf("destin is %s \n", buffer);
   destin = fopen( buffer,"w");
   if (destin == NULL)
   {  printf("%s Unable to open \n", buffer);
      exit(0);
   }
   rows = ReadEntry(source);          /* read input file */
   printf("m = %i \n", rows);
   Percent(rows);                     /* convert data to % change */

   for(i = 0; i < rows; i++)          /* print the file */
   {
      for(j = 0; j < COLMAX; j++)
      {
            printf("%s  %.3f \n", &date[i][j], a[i][j]);
      }
   }
   WriteEntry(destin, rows);                    /* write output file */
   fcloseall();
}
```

Build.C

```
/*************************************************************************
*  PROJECT:          CS 298
*  FILE:             build.c
*  PURPOSE:          Reads many files <name>.dat, and combines them into a
*                    table, and writes to a file ruff.tbl. The file names
*                    <name> are input from the command line. The .dat file
*                    is generated by the program data.exe. The format of file
*                    ruff.tbl is compatible with the DataLogic program.
*  VERSION:          1.0
*  LANGUAGE:         Borland C++ 4.02
*  TARGET:           IBM PC
*  PROGRAMMER:       Joe Tremba
*  UPDATE HISTORY:   8/14/96 original release
*                    9/3/96  added date check for entrys in ReadEntry.
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#define ROWMAX 2000      /* max number of rows */
#define ARRAYSIZE 20     /* size of array to store files */

/************ Function Prototypes ***********************************/
void Initialize(void);
int ReadEntry(FILE *, int);
int WriteEntry(FILE *fp, int, int);
void MaxMin (int,int);
void pause(void);

float a[ROWMAX][ARRAYSIZE];   /* array to store data */
char date[ROWMAX][ARRAYSIZE]; /* array to store entry date */

/*============================================================================
Initialize - puts the number 2000 in all array entries.
input: none
output: none
functions called: none
============================================================================*/
void Initialize()
{
   int i,j;

   for(i = 0; i < ROWMAX; i++)
   {
      for(j = 0; j < ARRAYSIZE; j++)
      {
         a[i][j] = 2000.0; /* set array values to 2000 */
         date[i][j] = NULL;   /* set date array to NULL */
      }
   }
}

/*============================================================================
ReadEntry - reads date and entries from file. Compares entry date with
            date array and errors if different. The first entry file is
            used to fill the date array.
input: pointer to file, number of column to place data
output: the number of rows in the file
functions called: none
============================================================================*/
int ReadEntry(FILE *fp, int j)
{
   int   x = 0,
         i = 0;
   char buffer[ARRAYSIZE];
   float temp;
   do
   {
      x = fscanf(fp, "%s%f", &buffer, &temp );
/*    printf("% i   %f\n", i, temp); */
      if(x != EOF && x != 0)
      {
         if(date[i][0] == NULL)     /* if date array empty, fill it */
            strcpy( &date[i][0], buffer);
         else if( strcmp( &date[i][0], buffer) != 0)/*check if dates match*/
         {
            printf("date error in row %i col %i, %s should be %s \n",
                    i, j, buffer, &date[i][0]);

            i = ROWMAX - 1;       /* force end of reading this file */
         }
         a[i][j] = temp;
         i++;
      }

   }while(x != EOF && x != 0 && i < ROWMAX);

   return i;
}
```

```
/*============================================================================
WriteEntry - writes array entries to file.
Input: File pointer, number of rows and columns
Output: 0 if success, -1 otherwise
Functions called: none
============================================================================*/
int WriteEntry(FILE *fp, int row, int col)
{
   int i,j;

   for(i = 0; i < row; i++)
   {
      for(j = 0; j < col; j++)
      {
         if(a[i][j] != 2000.)
            fprintf(fp, "%.3f ", a[i,j]);
         else
         {  printf("error: row %i col %i in array contains 2000\n",i,j);
            exit(0);
         }
      }
      fputc( '\n', fp );                /* add CR at end of line */
   }
   fputc(EOF, fp);
   return 0;
}

/*============================================================================
MaxMin - finds the maximum and minimum values in each column.
Input: number of rows and columns
Output: file containing values
Functions called: none
============================================================================*/
void MaxMin (int rows, int cols)
{
   int i, j, x;
   int max[ARRAYSIZE], min[ARRAYSIZE];
   FILE *fptr;

   for(j = 0; j < cols; j++)
   {
      max[j] = a[0][j] + 0.5;      /* set max, min to first cell*/
      min[j] = a[0][j] + 0.5;
      for(i = 1; i < rows; i++)
      {
         x = a[i][j] + 0.5;
         if( x > max[j])
            max[j] = x;
         else if( x < min[j])
            min[j] = x;
      }
   }
   fptr = fopen("ruff.txt","w");
   fprintf(fptr, "col\tmin\tmax\n");
   printf("col\tmin\tmax\n");
   for( j = 0; j < cols; j++)
   {
      fprintf(fptr, "%i\t%i\t%i\n", j, min[j],max[j]);
      printf("%i\t%i\t%i\n", j, min[j],max[j]);
   }
}

/*============================================================================
pause - stops the program until a keyboard character is entered.
Input: none
Output: none
Functions called: none
============================================================================*/
void pause ()
{
   getchar();
```

```
/*-------------------------------------------------------------------------*/
void main(int argc, char *argv[])
{
   FILE *source[ARRAYSIZE], *destin;
   int i, j, x[ARRAYSIZE], y;
   char buffer[14];

   if (argc < 2)
   {  printf("Usage: data <filename>");
      exit(0);
   }
   y = argc - 1;
   Initialize();
   for(j = 0; j < y; j++)
   {
      strcpy( buffer,argv[j+1] );
      strcat( buffer,".dat");
      printf("source is %s \n", buffer);
      source[j] = fopen( buffer,"r");
      if (source[j] == NULL)
      {  printf("%s File not found\n", buffer);
         exit(0);
      }
      x[j] = ReadEntry(source[j], j);  /* read input file */
      if(x[j] == 0)
      {
         printf("error: no data read from file %s \n", buffer);
         exit(0);
      }
   }
   for(j = 0; j < y; j++)
   {
      printf("col %i has %i rows\n", j, x[j]);
   }
   printf("destin is ruff.tbl\n");
   destin = fopen( "ruff.tbl","w");
   if (destin == NULL)
   {  printf("Unable to open ruff.tbl\n");
      exit(0);
   }
   WriteEntry(destin, x[0], y);        /* write output file */
   printf("ruff.tbl has %i rows and %i columns \n", x[0], y);
   fcloseall();

}
```

Bldmonth.C

```
/*************************************************************************
*  PROJECT:          CS 298
*  FILE:             bldmonth.c
*  PURPOSE:          Reads many files <name>.dat, and combines them into a
*                    table, and writes to a file ruff.tbl. The file names
*                    <name> are input from the command line. The .dat file
*                    is generated by the program datmonth.exe. The format of file
*                    ruff.tbl is compatible with the DataLogic program.
*  VERSION:          1.0
*  LANGUAGE:         Borland C++ 4.02
*  TARGET:           IBM PC
*  PROGRAMMER:       Joe Tremba
*  UPDATE HISTORY:   9/14/96 original release
*                    9/3/96  added date check for entrys in ReadEntry.
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#define ROWMAX 2000      /* max number of rows */
#define ARRAYSIZE 20     /* size of array to store files */

/*********** Function Prototypes **********************************/
void Initialize(void);
int ReadEntry(FILE *, int);
int WriteEntry(FILE *fp, int, int);
void MaxMin (int,int);
void pause(void);

float a[ROWMAX][ARRAYSIZE];   /* array to store data */
char date[ROWMAX][ARRAYSIZE]; /* array to store entry date */

/*=========================================================================
Initialize - puts the number 2000 in all array entries.
input: none
output: none
functions called: none
=========================================================================*/
void Initialize()
{
   int i,j;

   for(i = 0; i < ROWMAX; i++)
   {
      for(j = 0; j < ARRAYSIZE; j++)
      {
         a[i][j] = 2000.0; /* set array values to 2000 */
         date[i][j] = NULL;   /* set date array to NULL */
      }
   }
}

/*=========================================================================
ReadEntry - reads date and entries from file. Compares entry date with
            date array and errors if different. The first entry file is
            used to fill the date array.
input: pointer to file, number of column to place data
output: the number of rows in the file
functions called: none
=========================================================================*/
int ReadEntry(FILE *fp, int j)
{
   int   x = 0,
         i = 0;
   char buffer[ARRAYSIZE];
   float temp;
   do
   {
      x = fscanf(fp, "%s%f", &buffer, &temp );
      if(x != EOF && x != 0)
      {
         if(date[i][0] == NULL)     /* if date array empty, fill it */
            strcpy( &date[i][0], buffer);
         else if( strcmp( &date[i][0], buffer) != 0)/*check if dates match*/
         {
            printf("date error in row %i col %i, %s should be %s \n",
                    i, j, buffer, &date[i][0]);

            i = ROWMAX - 1;      /* force end of reading this file */
         }
         a[i][j] = temp;
         i++;
      }

   }while(x != EOF && x != 0 && i < ROWMAX);

   return i;
}

/*=========================================================================
```

```
WriteEntry - writes array entries to file.
input: File pointer, number of rows and columns
output: 0 if success, -1 otherwise
functions called: none
============================================================================*/
int WriteEntry(FILE *fp, int row, int col)
{
   int i,j;

   for(i = 0; i < row; i++)
   {
      for(j = 0; j < col; j++)
      {
         if(a[i][j] != 2000.)
            fprintf(fp, "%.3f ", a[i][j]);
         else
         {  printf("error: row %i col %i in array contains 2000\n",i,j);
            exit(0);
         }
      }
      fputc( '\n', fp );                /* add CR at end of line */
   }
   fputc(EOF, fp);
   return 0;
}

/*============================================================================
MaxMin - finds the maximum and minimum values in each column.
input: number of rows and columns
output: file containing values
functions called: none
============================================================================*/
void MaxMin (int rows, int cols)
{
   int i, j, x;
   int max[ARRAYSIZE], min[ARRAYSIZE];
   FILE *fptr;

   for(j = 0; j < cols; j++)
   {
      max[j] = a[0][j] + 0.5;      /* set max, min to first cell*/
      min[j] = a[0][j] + 0.5;
      for(i = 1; i < rows; i++)
      {
         x = a[i][j] + 0.5;
         if( x > max[j])
            max[j] = x;
         else if( x < min[j])
            min[j] = x;
      }
   }
   fptr = fopen("ruff.txt","w");
   fprintf(fptr, "col\tmin\tmax\n");
   printf("col\tmin\tmax\n");
   for( j = 0; j < cols; j++)
   {
      fprintf(fptr, "%i\t%i\t%i\n", j, min[j],max[j]);
      printf("%i\t%i\t%i\n", j, min[j],max[j]);
   }
}

/*============================================================================
pause - stops the program until a keyboard character is entered.
input: none
output: none
functions called: none
============================================================================*/
void pause ()
{
   getchar();
}
```

```
/*--------------------------------------------------------------------------*/
void main(int argc, char *argv[])
{
   FILE *source[ARRAYSIZE], *destin;
   int i, j, x[ARRAYSIZE], y;
   char buffer[14];

   if (argc < 2)
   {  printf("Usage: data <filename>");
      exit(0);
   }
   y = argc - 1;
   Initialize();
   for(j = 0; j < y; j++)
   {
      strcpy( buffer,argv[j+1] );
      strcat( buffer,".dat");
      printf("source is %s \n", buffer);
      source[j] = fopen( buffer,"r");
      if (source[j] == NULL)
      {  printf("%s File not found\n", buffer);
         exit(0);
      }
      x[j] = ReadEntry(source[j], j);  /* read input file */
      if(x[j] == 0)
      {
         printf("error: no data read from file %s \n", buffer);
         exit(0);
      }
   }
   for(j = 0; j < y; j++)
   {
      printf("col %i has %i rows\n", j, x[j]);
   }
   printf("destin is ruff.tbl\n");
   destin = fopen( "ruff.tbl","w");
   if (destin == NULL)
   {  printf("Unable to open ruff.tbl\n");
      exit(0);
   }
   MaxMin(x[0], y);
   WriteEntry(destin, x[0], y);        /* write output file */
   printf("ruff.tbl has %i rows and %i columns \n", x[0], y);
   fcloseall();

}
```

RufAver.C

```
/*****************************************************************************
*  PROJECT:           CS 298
*  FILE:              rufaver.c
*  PURPOSE:           Reads a file into an array. Then reproduces the array
*                     six times. Each of the six reproductions is formed by
*                     averaging the 5, 10, 15, 20, 25, and 30 previous rows
*                     respectively. The input file is rufruf.tbl. The output
*                     file is rufaver.tbl
*  VERSION:           1.0
*  LANGUAGE:          Borland C++ 4.02
*  TARGET:            IBM PC
*  PROGRAMMER:        Joe Tremba
*  UPDATE HISTORY:    7/31/96 original release
*                     4/12/97 modified fprintf format in MinMax. Added start
*                             row paramater in MinMax and WriteEntry.
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#define ROWMAX 1518
#define COLMAX 78
#define START  30

/************ Function Prototypes ************************************/
void Initialize(void);
void pause(void);
int ReadEntry(FILE *fp,int,int);
int WriteEntry(FILE *fp,int,int,int);
void MaxMin (int,int,int);
void Decision (int,int);
int Average(int,int,int,int);
int AverageTable(int,int);

float a[ROWMAX][COLMAX];

/*============================================================================
Initialize - puts the number 2000 in all array entries.
Input: none
Output: none
Functions called: none
============================================================================*/
void Initialize()
{
   int i,j;

   for(i = 0; i < ROWMAX; i++)
   {
      for(j = 0; j < COLMAX; j++)
      {
            a[i][j] = 2000.0;
      }
   }
}

/*============================================================================
ReadEntry - reads entries from file.
Input: pointer to file, number of rows and columns in table
Output: number of rows read
Functions called: none
============================================================================*/
int ReadEntry(FILE *fp, int rows, int cols)
{
   int   x = 0,
         i = 0,
         j;
   float temp;
   do
   {
      for(j = 0; j < cols; j++)
      {
         x = fscanf(fp, "%f", &temp );
         if(x != EOF && x != 0)
         {
            a[i][j] = temp;
         }
      }
      i++;
   }while(x != EOF && x != 0 && i < rows);

   return i;
}

/*============================================================================
WriteEntry - writes floating point array entries to file as rounded
             integers. A NULL character is added at the start of each entry.
Input: File pointer, number of rows and columns, starting row to print
Output: 0 if success, -1 otherwise
Functions called: none
============================================================================*/
```

```
int WriteEntry(FILE *fp, int rows, int cols, int start)
{
   int i,j,temp;
   float    x = 0;

   for(i = start; i < rows; i++)
   {
      j = 0;
      if(a[i][j] == 2000.)
         break;
      for(j = 0; j < cols; j++)
      {
         if(a[i][j] != 2000.)
         {
            fputc( NULL, fp);          /* insert a leading NULL */
            if( a[i][j] < 0 )
               x =  a[i][j] + 0.5;
            else
               x =  a[i][j] - 0.5;
            temp = x;
            fprintf(fp, "%-4i", temp);
         }
         else
         {  printf("error: column in array contains 2000\n");

         }
      }
      fputc( '\n', fp );               /* add CR at end of line */
   }
   fputc(EOF, fp);
   return 1;
}

/*============================================================================
MaxMin - finds the maximum and minimum values in each column and write to
         file rufaver.txt.
input: number of rows and columns, starting row
output: file containing values
functions called: none
============================================================================*/
void MaxMin (int rows, int cols, int start)
{
   int i, j, x;
   int max[COLMAX], min[COLMAX];
   FILE *fptr;

   for(j = 0; j < cols; j++)
   {
      if( a[0][j] < 0)
      {
         max[j] = a[0][j] + 0.5;      /* set max, min to first cell*/
         min[j] = a[0][j] + 0.5;
      }
      else
      {
         max[j] = a[0][j] - 0.5;      /* set max, min to first cell*/
         min[j] = a[0][j] - 0.5;
      }
      for(i = start; i < rows; i++)
      {
         if(a[i][j] < 0)
            x = a[i][j] + 0.5;
         else
            x = a[i][j] - 0.5;
         if( x > max[j])
            max[j] = x;
         else if( x < min[j])
            min[j] = x;
      }
   }
   fptr = fopen("rufaver.txt","w");
```

```
   fprintf(fptr, "col min max\n");
   printf("col min max\n");
   for( j = 0; j < cols; j++)
   {
      fprintf(fptr, "%-4i%-4i%-4i\n", j, min[j],max[j]);
      printf("%-4i%-4i%-4i\n", j, min[j],max[j]);
   }
}

/*============================================================================
Decision - adjusts the category ranges for decision attribute
           algorithm:   if value < -0.5 then d = -1
                        if -0.5 <= value < 0.5 then d = 0
                        if value >= 0.5 then d = 1
input: number of rows, column of decision attribute
output: nothing
functions called: none
============================================================================*/
void Decision (int rows, int col)
{
   int i;
   float x;

   for(i = 0; i < rows; i++)
   {
      x = a[i][col] + 0.5;
      if( x < 0.0 )
         a[i][col] = -1.0;
      else if( x < 1.0 )
         a[i][col] = -0.2;
      else
         a[i][col] = 0.8;
   }
}

/*============================================================================
Average - averages the data of previous rows. Also shifts the last column.
input: number of rows, number of columns, start of summed column, number
       of rows to sum
output: 0 if success, -1 otherwise
functions called: none
============================================================================*/
int Average(int rows, int delta, int to_start, int limit)
{
   int i,j, m;
   float average, sum;

   if(to_start + delta >= COLMAX)   /* check if array size is OK */
   {
      printf("error: array size too small in Average");
      exit(0);
   }
   for(i = rows - 1; i >0; i--)     /* start with last row */
   {
      for(j = 0; j < delta; j++) /* for column to be summed */
      {
         sum = 0.0;                /* initialize sum variable */
         for(m = 0; m < limit; m++) /* sum all rows */
         {
            if( i - m >= 0)         /* limit check on row */
            {
               sum = sum + a[i-m][j];
            }
         }
         average = sum / (1.0 * limit );  /* find average */
         a[i][j+to_start] = average;
         /* insert value from one row earlier */
      }
   }
   for(j = 0; j < delta; j++) /* for column to be summed */
   {
```

```
            a[0][j+to_start] = a[0][j];    /* keep first row the same */
    }
    return 0;
}

/*============================================================================
pause - stops the program until a keyboard character is entered.
input: none
output: none
functions called: none
============================================================================*/
void pause ()
{
    getchar();
}

/*============================================================================
AverageTable - averages the table six times.
input: number of rows and columns in original table
output: number of columns in new table
functions called: none
============================================================================*/
int AverageTable(int rows,int cols)
{
    FILE *source, *destin;
    int x,y;

    Initialize();
    source = fopen("ruff.tbl","r");
    if (source == NULL)
    {  printf("File ruff.tbl not found\n");
       exit(0);
    }
    destin = fopen("rufaver.tbl","w");
    if (destin == NULL)
    {  printf("Unable to open rufaver.tbl\n");
       exit(0);
    }
    ReadEntry(source, rows, cols );      /* read input file */
    x = cols;
    Average(rows,x,x,5);                        /* averages entries */
    Average(rows,x,2*x,10);
    Average(rows,x,3*x,15);
    Average(rows,x,4*x,20);
    Average(rows,x,5*x,25);
    Average(rows,x,6*x,30);
    y = 7 * x;
    Decision(rows, x - 1);
    WriteEntry(destin,rows,y, START);                        /* write output file */
    MaxMin(rows,y, START);
    fcloseall();
    return y;
}

/*----------------------------------------------------------------------------*/
void main( int argc, char *argv[])
{
    int rows, cols,x;

    if (argc != 3)
    {  printf("Usage: rufaver <rows> <cols> \n");
       exit(0);
    }
    rows = atoi(argv[1]);
    cols = atoi(argv[2]);
    if(cols == 0 || rows == 0)
    {
       printf("error: rows or cols = 0\n");
       exit(0);
    }
    x = AverageTable(rows,cols);
```

```
   printf("averaged table has %i rows and %i columns \n", rows - START, x);
}
```

mRufAver.C

```
/*****************************************************************************
*  PROJECT:          CS 298
*  FILE:             mrufaver.c
*  PURPOSE:          Reads a file into an array. Then reproduces the array
*                    six times. Each of the six reproductions is formed by
*                    averaging the 1, 2, 3, 4, 5, and 6 previous rows (months)
*                    respectively. The input file is ruff.tbl. The output
*                    file is mrufaver.tbl
*  VERSION:          1.0
*  LANGUAGE:         Borland C++ 4.02
*  TARGET:           IBM PC
*  PROGRAMMER:       Joe Tremba
*  UPDATE HISTORY:   7/31/96 original release
*                    4/12/97 modified fprintf format in MinMax. Added start
*                            row paramater in MinMax and WriteEntry.
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ROWMAX 1518
#define COLMAX 78
#define START  6

/*********** Function Prototypes **********************************/
void Initialize(void);
void pause(void);
int ReadEntry(FILE *fp,int,int);
int WriteEntry(FILE *fp,int,int,int);
void MaxMin (int,int,int);
void Decision (int,int);
int Average(int,int,int,int);
int AverageTable(int,int);

float a[ROWMAX][COLMAX];

/*===========================================================================
Initialize - puts the number 2000 in all array entries.
input: none
output: none
functions called: none
===========================================================================*/
void Initialize()
{
   int i,j;

   for(i = 0; i < ROWMAX; i++)
   {
      for(j = 0; j < COLMAX; j++)
      {
            a[i][j] = 2000.0;
      }
   }
}

/*===========================================================================
ReadEntry - reads entries from file.
input: pointer to file, number of rows and columns in table
output: number of rows read
functions called: none
===========================================================================*/
int ReadEntry(FILE *fp, int rows, int cols)
{
   int   x = 0,
         i = 0,
```

```
        };
    float temp;
    do
    {
       for(j = 0; j < cols; j++)
       {
          x = fscanf(fp, "%f", &temp );
          if(x != EOF && x != 0)
          {
             a[i][j] = temp;
          }
       }
       i++;
    }while(x != EOF && x != 0 && i < rows);

    return i;
}

/*============================================================================
WriteEntry - writes floating point array entries to file as rounded
             integers. A NULL character is added at the start of each entry.
input: File pointer, number of rows and columns, starting row to print
output: 0 if success, -1 otherwise
functions called: none
============================================================================*/
int WriteEntry(FILE *fp, int rows, int cols, int start)
{
    int i,j,temp = 0;
    float    x = 0;

    for(i = start; i < rows; i++)
    {
       j = 0;
       if(a[i][j] == 2000.)
          break;
       for(j = 0; j < cols; j++)
       {
          if(a[i][j] != 2000.)
          {
             fputc( NULL, fp);          /* insert a leading NULL */
             if( a[i][j] > 0 )
                x =  a[i][j] + 0.5;
             else
                x =  a[i][j] - 0.5;
             temp = x;
             fprintf(fp, "%-4i", temp);
          }
          else
          {  printf("error: column in array contains 2000\n");

          }
       }
       fputc( '\n', fp );             /* add CR at end of line */
    }
    fputc(EOF, fp);
    return 1;
}

/*============================================================================
MaxMin - finds the maximum and minimum values in each column and write to
         file rufaver.txt.
input: number of rows and columns, starting row
output: file containing values
functions called: none
============================================================================*/
void MaxMin (int rows, int cols, int start)
{
    int i, j, x;
    int max[COLMAX], min[COLMAX];
    FILE *fptr;
```

```
   for(j = 0; j < cols; j++)
   {
      if( a[0][j] < 0)
      {
         max[j] = a[0][j] + 0.5;      /* set max, min to first cell*/
         min[j] = a[0][j] + 0.5;
      }
      else
      {
         max[j] = a[0][j] - 0.5;      /* set max, min to first cell*/
         min[j] = a[0][j] - 0.5;
      }
      for(i = start; i < rows; i++)
      {
         if(a[i][j] < 0)
            x = a[i][j] + 0.5;
         else
            x = a[i][j] - 0.5;
         if( x > max[j])
            max[j] = x;
         else if( x < min[j])
            min[j] = x;
      }
   }
   fptr = fopen("mrufaver.txt","w");
   fprintf(fptr, "col min max\n");
   printf("col min max\n");
   for( j = 0; j < cols; j++)
   {
      fprintf(fptr, "%-4i%-4i%-4i\n", j, min[j],max[j]);
      printf("%-4i%-4i%-4i\n", j, min[j],max[j]);
   }
}

/*=========================================================================
Decision - adjusts the category ranges for decision attribute
           algorithm:   if value < -0.5 then d = -1
                        if  0.5 <= value < 0.5 then d = 0
                        if value >= 0.5 then d = 1
input: number of rows, column of decision attribute
output: nothing
functions called: none
=========================================================================*/
void Decision (int rows, int col)
{
   int i;
   float x;

   for(i = 0; i < rows; i++)
   {
      x = a[i][col] + 0.5;
      if( x < 0.0 )
         a[i][col] = -1.0;
      else if( x < 1.0 )
         a[i][col] = -0.2;
      else
         a[i][col] = 0.8;
   }
}

/*-------------------------------------------------------------------------
Average - averages the data of previous rows. Also shifts the last column.
input: number of rows, number of columns, start of summed column, number
       of rows to sum
output: 0 if success, -1 otherwise
functions called: none
=========================================================================*/
int Average(int rows, int delta, int to_start, int limit)
{
   int i,j, m;
   float average, sum;
```

```
    if(to_start + delta >= COLMAX)   /* check if array size is OK */
    {
       printf("error: array size too small in Average");
       exit(0);
    }
    for(i = rows - 1; i >0; i--)     /* start with last row */
    {
       for(j = 0; j < delta; j++) /* for column to be summed */
       {
          sum = 0.0;                /* initialize sum variable */
          for(m = 0; m < limit; m++) /* sum all rows */
          {
             if( i - m >= 0)        /* limit check on row */
             {
                sum = sum + a[i-m][j];
             }
          }
          average = sum / (1.0 * limit );  /* find average */
          a[i][j+to_start] = average;
          /* insert value from one row earlier */
       }
    }
    for(j = 0; j < delta; j++) /* for column to be summed */
    {
          a[0][j+to_start] = a[0][j];   /* keep first row the same */
    }
    return 0;
}

/*============================================================================
pause - stops the program until a keyboard character is entered.
input: none
output: none
functions called: none
============================================================================*/
void pause ()
{
   getchar();
}

/*============================================================================
AverageTable - averages the table six times.
input: number of rows and columns in original table
output: number of columns in new table
functions called: none
============================================================================*/
int AverageTable(int rows,int cols)
{
   FILE *source, *destin;
   int x,y;

   Initialize();
   source = fopen("ruff.tbl","r");
   if (source == NULL)
   {  printf("File ruff.tbl not found\n");
      exit(0);
   }
   destin = fopen("mrufaver.tbl","w");
   if (destin == NULL)
   {  printf("Unable to open rufaver.tbl\n");
      exit(0);
   }
   ReadEntry(source, rows, cols );     /* read input file */
   x = cols;
   Average(rows,x,x,1);                      /* averages entries */
   Average(rows,x,2*x,2);
   Average(rows,x,3*x,3);
   Average(rows,x,4*x,4);
   Average(rows,x,5*x,5);
   Average(rows,x,6*x,6);
```

```
   y = 7 * x;
   Decision(rows, x - 1);
   WriteEntry(destin,rows,y, START);                    /* write output file */
   MaxMin(rows,y, START);
   fcloseall();
   return y;
}

/*-------------------------------------------------------------------------*/
void main( int argc, char *argv[])
{
   int rows, cols,x;

   if (argc != 3)
   {  printf("Usage: rufaver <rows> <cols> \n");
      exit(0);
   }
   rows = atoi(argv[1]);
   cols = atoi(argv[2]);
   if(cols == 0 || rows == 0)
   {
      printf("error: rows or cols = 0\n");
      exit(0);
   }
   x = AverageTable(rows,cols);
   printf("averaged table has %i rows and %i columns \n", rows - START, x);
}
```

RufDelay.C

```
/*************************************************************************
*  PROJECT:          CS 298
*  FILE:             rufdelay.c
*  PURPOSE:          Reads a file into an array. Then reproduces the array
*                    six times. Each of the six reproductions is delayed by
*                    one row from the previous reproduction. The input file
*                    is ruff.tbl. The output file is rufdelay.tbl
*  VERSION:          1.0
*  LANGUAGE:         Borland C++ 4.02
*  TARGET:           IBM PC
*  PROGRAMMER:       Joe Tremba
*  UPDATE HISTORY:   7/31/96 original release
*                    9/25/96 delay decision attribute also. Columns = 77
*                    4/12/97 modified fprintf format in MinMax
*                    4/12/97 modified fprintf format in MinMax. Added start
*                            row paramater in MinMax and WriteEntry.
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ROWMAX 1518
#define COLMAX 78
#define START  6

/********** Function Prototypes **********************************/
void Initialize(void);
int ReadEntry(FILE *fp,int,int);
int WriteEntry(FILE *fp,int,int,int);
void MaxMin (int,int,int);
void Decision (int,int);
int OneDelay(int,int,int);
void pause(void);
int DelayTable(int,int);

float a[ROWMAX][COLMAX];

/*=========================================================================
Initialize - puts the number 2000 in all array entries.
input: none
```

```
Output: none
Functions called: none
================================================================================*/
void Initialize()
{
   int i,j;

   for(i = 0; i < ROWMAX; i++)
   {
      for(j = 0; j < COLMAX; j++)
      {
            a[i][j] = 2000.0;
      }
   }
}

/*================================================================================
ReadEntry - reads entries from file.
Input: pointer to file, number of rows and columns in table
Output: number of rows read
Functions called: none
================================================================================*/
int ReadEntry(FILE *fp, int rows, int cols)
{
   int   x = 0,
         i = 0,
         j;
   float temp;
   do
   {
      for(j = 0; j < cols; j++)
      {
         x = fscanf(fp, "%f", &temp );
         if(x != EOF && x != 0)
         {
            a[i][j] = temp;
         }
      }
         i++;
   }while(x != EOF && x != 0 && i < rows);

   return i;
}

/*================================================================================
WriteEntry - writes array entries to file. Rounds entries to nearest
             integer. Does not Scales entries by 1/6.
Input: File pointer, number of rows and columns, starting row
Output: 0 if success, -1 otherwise
Functions called: none
================================================================================*/
int WriteEntry(FILE *fp, int rows, int cols, int start)
{
   int i,j,temp;
   float    x = 0;

   for(i = start; i < rows; i++)
   {
      j = 0;
      if(a[i][j] == 2000.)
         break;
      for(j = 0; j < cols; j++)
      {
         if(a[i][j] != 2000.)
         {
            fputc( NULL, fp);          /* insert a leading NULL */
            if( a[i][j] > 0 )
               x =  a[i][j] + 0.5;
            else
               x =  a[i][j] - 0.5;
            temp = x;
```

```
            fprintf(fp, "%-4i", temp);
         }
         else
         {  printf("error: row %i column %i in array contains 2000\n", i,j);

         }
      }
      fputc( '\n', fp );                /* add CR at end of line */
   }
   fputc(EOF, fp);
   return 1;
}

/*============================================================================
MaxMin - finds the maximum and minimum values in each column and write to
         file rufaver.txt.
input: number of rows and columns, starting row
output: file containing values
functions called: none
============================================================================*/
void MaxMin (int rows, int cols, int start)
{
   int i, j, x;
   int max[COLMAX], min[COLMAX];
   FILE *fptr;

   for(j = 0; j < cols; j++)
   {
      if( a[0][j] > 0)
      {
         max[j] = a[0][j] + 0.5;      /* set max, min to first cell*/
         min[j] = a[0][j] + 0.5;
      }
      else
      {
         max[j] = a[0][j] - 0.5;      /* set max, min to first cell*/
         min[j] = a[0][j] - 0.5;
      }
      for(i = start; i < rows; i++)
      {
         if(a[i][j] > 0)
            x = a[i][j] + 0.5;
         else
            x = a[i][j] - 0.5;
         if( x > max[j])
            max[j] = x;
         else if( x < min[j])
            min[j] = x;
      }
   }
   fptr = fopen("rufdelay.txt","w");
   fprintf(fptr, "col min max\n");
   printf("col min max\n");
   for( j = 0; j < cols; j++)
   {
      printf("%-4i%-4i%-4i\n", j, min[j],max[j]);
      fprintf(fptr, "%-4i%-4i%-4i\n", j, min[j],max[j]);
   }
}

/*============================================================================
Decision - adjusts the category ranges for decision attribute
           algorithm:   if value < -0.5 then d = -1
                        if -0.5 <= value < 0.5 then d = 0
                        if value >= 0.5 then d = 1
input: number of rows, column of decision attribute
output: nothing
functions called: none
============================================================================*/
void Decision (int rows, int col)
{
```

```
    int i;
    float x;

    for(i = 0; i < rows; i++)
    {
       x = a[i][col] + 0.5;
       if( x < 0.0 )
          a[i][col] = -1.0;
       else if( x < 1.0 )
          a[i][col] = -0.2;
       else
          a[i][col] = 0.8;
    }
}

/*=========================================================================
OneDelay - delays the data by one row. Also shifts the last column.
input: number of rows, starting column to delay, start of delayed column
output: 0 if success, -1 otherwise
functions called: none
=========================================================================*/
int OneDelay(int rows, int from_start,int to_start)
{
    int i,j, delta;

    delta = to_start - from_start;    /* determine number of columns to delay */
    if (from_start == 0)              /* adjust column for first delay */
       delta++;
    if(to_start + delta >= COLMAX)    /* check if array size is OK */
    {
       printf("error: array size too small in OneDelay");
       exit(0);
    }
    for(i = rows - 1; i >0; i--)      /* start with last row */
    {
       for(j = 0; j < delta; j++) /* for column to be delayed */
       {
          a[i][j+to_start] = a[i-1][j+from_start];
          /* insert value from one row earlier */
       }
    }
    for(j = 0; j < delta; j++) /* for column to be delayed */
    {
          a[0][j+to_start] = a[0][j+from_start]; /* keep first row the same */
    }
    return 0;
}

/*=========================================================================
pause - stops the program until a keyboard character is entered.
input: none
output: none
functions called: none
=========================================================================*/
void pause ()
{
    getchar();
}

/*=========================================================================
DelayTable - delays the table six times.
input: number of rows and columns in original table
output: number of columns in new table
functions called: none
=========================================================================*/
int DelayTable(int rows,int cols)
{
    FILE *source, *destin;
    int x,y;

    Initialize();
```

```
   source = fopen("ruff.tbl","r");
   if (source == NULL)
   {  printf("File ruff.tbl not found\n");
      exit(0);
   }
   destin = fopen("rufdelay.tbl","w");
   if (destin == NULL)
   {  printf("Unable to open rufdelay.tbl\n");
      exit(0);
   }
   ReadEntry(source, rows, cols );      /* read input file */
   x = cols;
   OneDelay(rows,0,x);                        /* delays entries one row */
   OneDelay(rows,x,2*x);
   OneDelay(rows,2*x,3*x);
   OneDelay(rows,3*x,4*x);
   OneDelay(rows,4*x,5*x);
   OneDelay(rows,5*x,6*x);
   y = 7 * x;
   Decision(rows, x - 1);
   WriteEntry(destin,rows,y,START);                    /* write output file */
   MaxMin(rows,y,START);
      fcloseall();
   return y;
}

/*-------------------------------------------------------------------------*/
void main( int argc, char *argv[])
{
   int rows, cols,x;

   if (argc != 3)
   {  printf("Usage: rufdelay <rows> <cols> \n");
      exit(0);
   }
   rows = atoi(argv[1]);
   cols = atoi(argv[2]);
   if(cols == 0 || rows == 0)
   {
      printf("error: rows or cols = 0\n");
      exit(0);
   }
   x = DelayTable(rows,cols);
   printf("delayed table has %i rows and %i columns \n", rows - START, x);
}
```

RufSum.C

```
/*************************************************************************
*  PROJECT:          CS 298
*  FILE:             rufsum.c
*  PURPOSE:          Reads a file into an array. Then reproduces the array
*                    six times. Each of the six reproductions is formed by
*                    summing the 2, 3, 4, 5, and 6 previous rows respectively.
*                    The input file is ruff.prn. The output file is
*                    rufsum.tbl
*  VERSION:          1.0
*  LANGUAGE:         Borland C++ 4.02
*  TARGET:           IBM PC
*  PROGRAMMER:       Joe Tremba
*  UPDATE HISTORY:   7/31/96 original release
*                    11/28/96 included the summing of the eleventh column
*                    4/12/97 modified fprintf format in MinMax. Added start
*                            row paramater in MinMax and WriteEntry.
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#define ROWMAX 1518
#define COLMAX 78
#define START  6

/*********** Function Prototypes **********************************/
void Initialize(void);
int ReadEntry(FILE *fp,int,int);
int WriteEntry(FILE *fp,int,int,int);
void MaxMin (int,int,int);
void Decision (int,int);
int Sum(int,int,int,int);
void pause(void);
int SumTable(int,int);

float a[ROWMAX][COLMAX];

/*============================================================================
Initialize - puts the number 2000 in all array entries.
input: none
output: none
functions called: none
============================================================================*/
void Initialize()
{
   int i,j;

   for(i = 0; i < ROWMAX; i++)
   {
      for(j = 0; j < COLMAX; j++)
      {
            a[i][j] = 2000.0;
      }
   }
}

/*============================================================================
ReadEntry - reads entries from file.
input: pointer to file, number of rows and columns in table
output: number of rows read
functions called: none
============================================================================*/
int ReadEntry(FILE *fp, int rows, int cols)
{
   int   x = 0,
         i = 0,
         j;
   float temp;
   do
   {
      for(j = 0; j < cols; j++)
      {
         x = fscanf(fp, "%f", &temp );
         if(x != EOF && x != 0)
         {
            a[i][j] = temp;
         }
      }
         i++;
   }while(x != EOF && x != 0 && i < rows);

   return i;
}

/*============================================================================
WriteEntry - writes array entries to file.
input: File pointer, number of rows and columns, starting row
output: 0 if success, -1 otherwise
functions called: none
============================================================================*/
int WriteEntry(FILE *fp, int rows, int cols, int start)
{
```

```
    int i,j,temp;
    float   x = 0;

    for(i = start; i < rows; i++)
    {
       j = 0;
       if(a[i][j] == 2000.)
          break;
       for(j = 0; j < cols; j++)
       {
          if(a[i][j] != 2000.)
          {
             fputc( NULL, fp);          /* insert a leading NULL */
             if( a[i][j] > 0 )
                x =  a[i][j] + 0.5;
             else
                x =  a[i][j] - 0.5;
             temp = x;
             fprintf(fp, "%-4i", temp);
          }
          else
          {  printf("error: column in array contains 2000\n");

          }
       }
       fputc( '\n', fp );              /* add CR at end of line */
    }
    fputc(EOF, fp);
    return i;
}

/*============================================================================
MaxMin - finds the maximum and minimum values in each column and write to
         file rufaver.txt.
input: number of rows and columns, starting row
output: file containing values
functions called: none
============================================================================*/
void MaxMin (int rows, int cols, int start)
{
    int i, j, x;
    int max[COLMAX], min[COLMAX];
    FILE *fptr;

    for(j = 0; j < cols; j++)
    {
       if( a[0][j] > 0)
       {
          max[j] = a[0][j] + 0.5;      /* set max, min to first cell*/
          min[j] = a[0][j] + 0.5;
       }
       else
       {
          max[j] = a[0][j] - 0.5;      /* set max, min to first cell*/
          min[j] = a[0][j] - 0.5;
       }
       for(i = start; i < rows; i++)
       {
          if(a[i][j] > 0)
             x = a[i][j] + 0.5;
          else
             x = a[i][j] - 0.5;
          if( x > max[j])
             max[j] = x;
          else if( x < min[j])
             min[j] = x;
       }
    }
    fptr = fopen("rufsum.txt","w");
    fprintf(fptr, "col min max\n");
    printf("col min max\n");
```

```
   for( j = 0; j < cols; j++)
   {
      fprintf(fptr, "%-4i%-4i%-4i\n", j, min[j],max[j]);
      printf("%-4i%-4i%-4i\n", j, min[j],max[j]);
   }
}

/*============================================================================
Decision - adjusts the category ranges for decision attribute
           algorithm:   if value < -0.5 then d = -1
                        if -0.5 <= value < 0.5 then d = 0
                        if value >= 0.5 then d = 1
input: number of rows, column of decision attribute
output: nothing
functions called: none
============================================================================*/
void Decision (int rows, int col)
{
   int i;
   float x;

   for(i = 0; i < rows; i++)
   {
      x = a[i][col] + 0.5;
      if( x < 0.0 )
         a[i][col] = -1.0;
      else if( x < 1.0 )
         a[i][col] = 0.0;
      else
         a[i][col] = 0.8;
   }
}

/*============================================================================
Sum - sums the data of previous rows.
input: number of rows, number of columns, start of summed column, number
       of rows to sum
output: 0 if success, -1 otherwise
functions called: none
============================================================================*/
int Sum(int rows, int delta, int to_start, int limit)
{
   int i,j, m;
   float sum;

   if(to_start + delta >= COLMAX)   /* check if array size is OK */
   {
      printf("error: array size too small in Sum");
      exit(0);
   }
   for(i = rows - 1; i > 0; i--)     /* start with last row */
   {
      for(j = 0; j < delta; j++)    /* for column to be summed */
      {
         sum = 0.0;                 /* initialize sum variable */
         for(m = 0; m < limit; m++) /* sum all rows */
         {
            if( i - m >= 0)         /* limit check on row */
            {
               sum = sum + a[i-m][j];
            }
         }
         a[i][j+to_start] = sum;    /* insert value */
      }
   }
   for(j = 0; j < delta; j++) /* for column to be summed */
   {
         a[0][j+to_start] = a[0][j];   /* keep first row the same */
   }
   return 0;
}
```

```
/*============================================================================
pause - stops the program until a keyboard character is entered.
input: none
output: none
functions called: none
============================================================================*/
void pause ()
{
   getchar();
}

/*============================================================================
SumTable - sums the table six times.
input: number of rows and columns in original table
output: number of columns in new table
functions called: none
============================================================================*/
int SumTable(int rows,int cols)
{
   FILE *source, *destin;
   int x,y;

   Initialize();
   source = fopen("ruff.tbl","r");
   if (source == NULL)
   {  printf("File ruff.tbl not found\n");
      exit(0);
   }
   destin = fopen("rufsum.tbl","w");
   if (destin == NULL)
   {  printf("Unable to open rufsum.tbl\n");
      exit(0);
   }
   ReadEntry(source, rows, cols );      /* read input file */
   x = cols;
   Sum(rows,x,x,1);                        /* sums entries one row */
   Sum(rows,x,2*x,2);
   Sum(rows,x,3*x,3);
   Sum(rows,x,4*x,4);
   Sum(rows,x,5*x,5);
   Sum(rows,x,6*x,6);
   y = 7 * x;
   Decision(rows, x - 1);
   WriteEntry(destin,rows,y,START);                    /* write output file */
   MaxMin(rows,y,START);
   fcloseall();
   return y;
}

/*----------------------------------------------------------------------------*/
void main( int argc, char *argv[])
{
   int rows, cols,x;

   if (argc != 3)
   {  printf("Usage: rufsum <rows> <cols> \n");
      exit(0);
   }
   rows = atoi(argv[1]);
   cols = atoi(argv[2]);
   if(cols == 0 || rows == 0)
   {
      printf("error: rows or cols = 0\n");
      exit(0);
   }
   x = SumTable(rows,cols);
   printf("summed table has %i rows and %i columns \n", rows - START, x);
}
```

Make_Typ.C

```
/*************************************************************************
*  PROJECT:          CS 298
*  FILE:             make_typ.c
*  PURPOSE:          Reads the file <name>.txt and <name>.typ file. Converts
*                    the max and min values into the <name>.typ file to those
*                    in the  <name>.txt file
*  VERSION:          1.0
*  LANGUAGE:         Borland C++ 4.02
*  TARGET:           IBM PC
*  PROGRAMMER:       Joe Tremba
*  UPDATE HISTORY:   9/5/96 original release
*                    6/16/97 corrected bug in ReadTyp that printed an extra
*                            line at end on file. Added EOF character.
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ROWMAX 2000      /* max number of rows */
#define COLMAX 78        /* max number of columns */
#define ARRAYSIZE 20     /* size of array to store files */

/********** Function Prototypes **********************************/
void Initialize(void);
int ReadMinMax(FILE *);
int ReadTyp(FILE *, FILE *, int);
void pause(void);

int min[COLMAX], max[COLMAX];     /* array to store min & max data */
char buffer[ARRAYSIZE]; /* array to store entry date */
char typ[ROWMAX][ARRAYSIZE];  /* array to store .typ file */

/*=========================================================================
Initialize - puts the number 2000 in all array entries.
input: none
output: none
functions called: none
=========================================================================*/
void Initialize()
{
   int i,j;

   for(i = 0; i < COLMAX; i++)
   {
      min[i] = NULL;
      max[i] = NULL;
   }
   for(j = 0; j < ARRAYSIZE; j++)
   {
        buffer[j] = NULL; /* set date array to NULL */
   }
   for(i = 0; i < ROWMAX; i++)
   {
      for (j = 0; j < ARRAYSIZE; j++)  /* set typ array to NULL */
      {
         typ[i][j] = NULL;
      }
   }
}

/*=========================================================================
ReadMinMax - reads min and max numbers from <name>.txt file and puts them
             in arrays.
input: pointer to file
output: the number of rows in the file
functions called: none
=========================================================================*/
int ReadMinMax(FILE *fp)
```

```
{
   int   x = 0,
         i = 0;
   char buffer[ARRAYSIZE];
   int temp, temp1, temp2;
   x = fscanf(fp, "%s%s%s", &buffer, &buffer, &buffer ); /* discard header */
   do
   {
      x = fscanf(fp, "%i%i%i", &temp, &temp1, &temp2 );
      if(x != EOF && x != 0)
      {
         if(temp != i)     /* if date array empty, fill it */
         {
            printf("col error in col %i, %i should be %i \n",
                    i, temp, i );

            i = ROWMAX;    /* force end of reading this file */
         }
         else
         {
            min[i] = temp1;
            max[i] = temp2;
            i++;
         }
      }
   }while(x != EOF && x != 0 && i < ROWMAX);

   return i;
}

/*============================================================================
ReadTyp - reads <name>.typ file, updates the min and max values for
          each column, and writes a new .typ file.
input: pointers to input and output files, number of min/max columns
output: the number of rows in the file
functions called: none
============================================================================*/
int ReadTyp(FILE *fin, FILE *fout, int col)
{
   int   x = 0, j;
   char buffer[ARRAYSIZE], buffer2[ARRAYSIZE];
   int ch, temp1, temp2, temp3;

   x = fscanf(fin, "%s", &buffer);
   while(strcmp(buffer, "II") != 0)
   {
      fprintf(fout, "%s ", buffer);
      ch = fgetc(fin);
      while(ch != '\n') /* advance to next line */
      {
         fputc(ch, fout);
         ch = fgetc(fin);
      }
      fputc(ch, fout);
      x = fscanf(fin, "%s", &buffer);  /* read first string of new line */
      if(x == EOF || x == 0)
      {
         printf("error: scanf failed \n");
         exit(0);
      }
   }
   ungetc(' ',fin);  /* restore II to input */
   ungetc('1',fin);
   ungetc('I', fin);
   for(j = 0; j < col; j++)
   {
      x = fscanf(fin, "%s %s %i %i %i", &buffer,&buffer2,&temp1,&temp2,&temp3);
      if(x != EOF && x != 0)
      {
         fprintf(fout,"%s %s %i %i %i \n",buffer,buffer2,min[j],max[j],
            abs(max[j]-min[j]) );
```

```
            printf("%s %s %i %i %i \n", buffer, buffer2, min[j], max[j],
               abs(max[j]-min[j]) );
         }
         else
         {
            printf("error: scanf failed \n");
            exit(0);
         }
      }
      x = fscanf(fin, "%s", &buffer);
      while( x != EOF && x != 0)
      {
         fprintf(fout, "%s \n", buffer);
         x = fscanf(fin, "%s", &buffer);
      }
      fputc(EOF, fout);
      return col;
}

/*============================================================================
pause - stops the program until a keyboard character is entered.
input: none
output: none
functions called: none
============================================================================*/
void pause ()
{
   getchar();
}

/*---------------------------------------------------- ----------------------*/
void main(int argc, char *argv[])
{
   FILE *txt, *source, *destin;
   int col, j, x;
   char buffer[ARRAYSIZE],
        buffer2[ARRAYSIZE],
        buffer3[ARRAYSIZE];

   if (argc != 2)
   {  printf("Usage: make_typ <filename>");
      exit(0);
   }

   Initialize();
   for(j = 0; j < 1; j++)
   {
      strcpy( buffer,argv[1] );
      strcpy( buffer2,argv[1] );
      strcat( buffer,".txt");
      printf("source is %s \n", buffer);
      txt = fopen( buffer,"r");
      if (txt == NULL)
      {  printf("%s File not found\n", buffer);
         exit(0);
      }
      col = ReadMinMax(txt);
      if(col == 0)
      {
         printf("error: no data read from file %s \n", buffer);
         exit(0);
      }
   }
   /* open source .typ file */
   strcpy( buffer2,argv[1] );
   strcat( buffer2,".ty");
   strcpy( buffer3, argv[1]);
   strcat(buffer3, ".typ");
   x = rename( buffer3, buffer2);
   if(x != 0)
   {
```

```
      printf("error: rename of %s to %s unsuccessful \n", buffer3,buffer2);
      exit(0);
   }
   printf("source is %s \n", buffer2);
   source = fopen( buffer2,"r");
   if (source == NULL)
   {  printf("%s File not found\n", buffer2);
      exit(0);
   }
   printf("destin is %s\n", buffer3);        /* open destin .typ file */
   destin = fopen( buffer3,"w");
   if (destin == NULL)
   {  printf("Unable to open file %s\n", buffer3);
      exit(0);
   }
   ReadTyp(source, destin, col);
   fcloseall();
   rename( buffer3, buffer2);
   printf("number of col is %i \n", col);

}
```

Appendix E - Batch Programs

Daily Batch Program

```
@echo off
rem ** Batch file to construct daily stock experiment
rem ** Joe Tremba 6/15/97
rem ** program output is appended to file stock.txt
@echo on
rem ** convert daily *.asc file to daily *.dat file
data dji >> stock.txt
data cpq >> stock.txt
data ibm >> stock.txt
data dec >> stock.txt
data intc >> stock.txt
data mot >> stock.txt
data txn >> stock.txt
data ese >> stock.txt
data ele >> stock.txt
data eee >> stock.txt
data amat >> stock.txt
rem ** build tables from daily *.dat files **
build dji cpq ibm dec intc mot txn ese ele eee amat >> stock.txt
rem ** make monthly tables and text files **
rufaver 1517 11 >> stock.txt
rufdelay 1517 11 >> stock.txt
rufsum 1517 11 >> stock.txt
rem ** make .typ files **
del *.ty >> stock.txt
make_typ rufaver >> stock.txt
make_typ rufdelay >> stock.txt
make_typ rufsum >> stock.txt
```

Monthly Batch Program

```
@echo off
rem ** Batch file to construct monthly stock experiment
rem ** Joe Tremba 6/15/97
rem ** program output is appended to file stockm.txt
@echo on
rem ** convert monthly *.asc file to monthly *.dat file
data siabb >> stockm.txt
data semibb >> stockm.txt
data semibbfe >> stockm.txt
rem ** convert daily *.asc file to monthly *.dat file **
datmonth amatm >> stockm.txt
rem ** build month tables from monthly *.dat files **
bldmonth siabb semibb semibbfe amatmm >> stockm.txt
rem ** make monthly tables and text files **
mrufaver 66 4 >> stockm.txt
```

```
rufdelay 66 4 >> stockm.txt
rufsum 66 4 >> stockm.txt
rem ** delete old files so rename will work **
del mrufdely.tbl >> stockm.txt
del mrufdely.txt >> stockm.txt
del mrufsum.tbl >> stockm.txt
del mrufsum.txt >> stockm.txt
rem ** rename table and text files to month files names **
ren rufdelay.tbl mrufdely.tbl >> stockm.txt
ren rufdelay.txt mrufdely.txt >> stockm.txt
ren rufsum.tbl mrufsum.tbl >> stockm.txt
ren rufsum.txt mrufsum.txt >> stockm.txt
rem ** make .typ files **
del *.ty >> stockm.txt
make_typ mrufaver >> stockm.txt
make_typ mrufdely >> stockm.txt
make_typ mrufsum >> stockm.txt
```